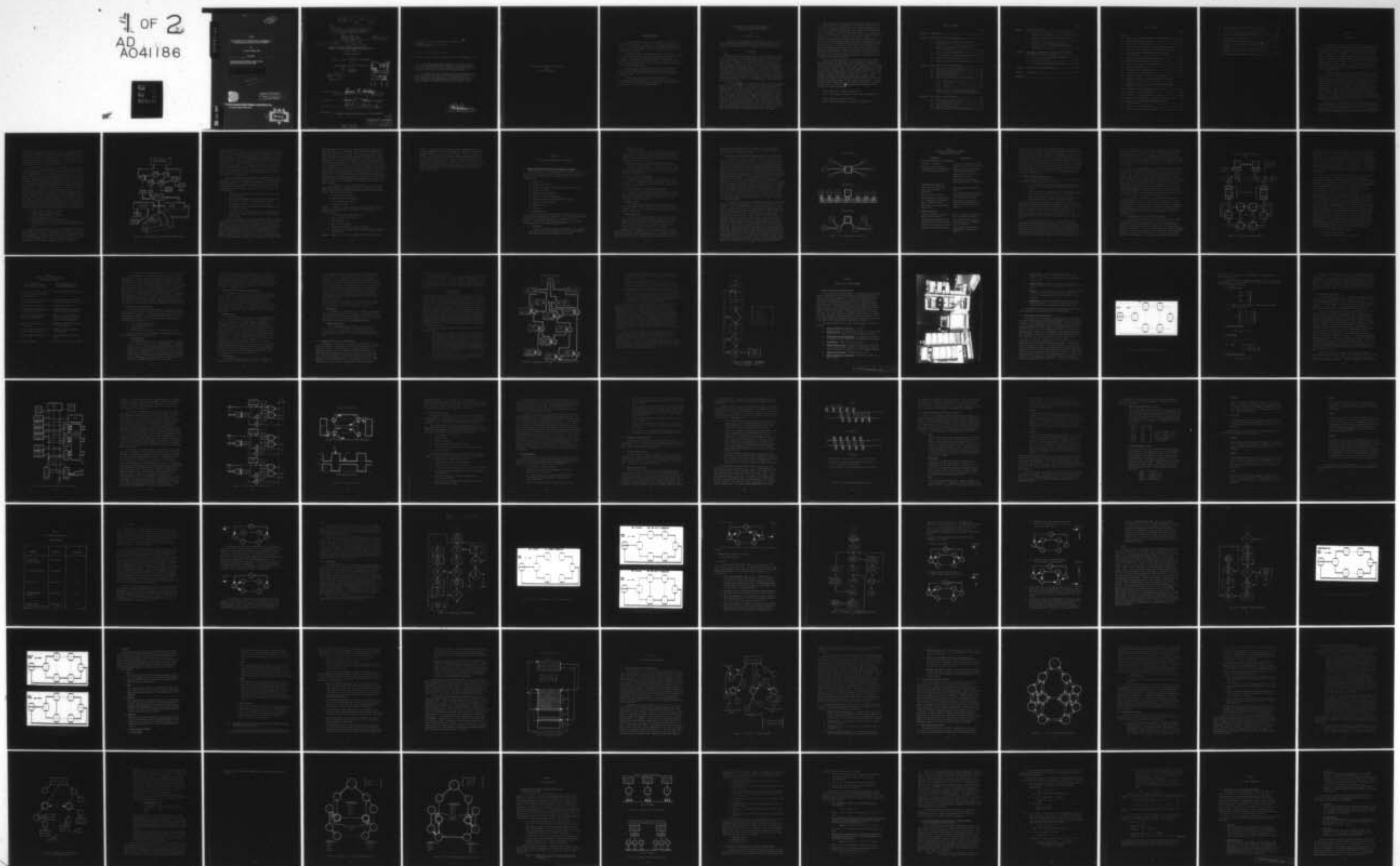
AD-A041 186 CHARLES STARK DRAPER LAB INC CAMBRIDGE MASS F/G 17/2
DEVELOPMENT OF A HIERARCHICAL, DYNAMICALLY RECONFIGURABLE INPUT--ETC(U)
JUN 77 J R SEELEY N00014-76-C-0502
UNCLASSIFIED T-644 NL

1 OF 2
AD
A041186



(12)
B.S.

T-644

DEVELOPMENT OF A HIERARCHICAL, DYNAMICALLY RECONFIGURABLE INPUT/OUTPUT NETWORK

by

Lt. James R. Seeley, USN

June, 1977

Electrical Engineer and Master of Science Thesis
Massachusetts Institute of Technology

Contract No. N00014-76-C-0502

Code 430C
NR 048-633

The Charles Stark Draper Laboratory, Inc.
Cambridge, Massachusetts 02139

Master's thesis,

DEVELOPMENT OF A HIERARCHICAL, DYNAMICALLY
RECONFIGURABLE INPUT/OUTPUT NETWORK.

by

Lt. James R. Seeley USN

S.B., United States Naval Academy (1972)

N00014-76-C-0502,
NSF-DCR-74-24116

T-644

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREES OF

ELECTRICAL ENGINEER

and

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1977

111 p.

Signature of Author James R. Seeley
Department of Electrical Engineering and Computer Science
May 12, 1977

Certified by Hoo-min D. Toong
Thesis Supervisor

Certified by Albert L. Hopkins Jr.
Charles Stark Draper Laboratory Supervisor

Accepted by
Chairman, Departmental Committee on Graduate Students

408386

Department of Electrical Engineering and Computer Science
Room 38-444
Massachusetts Institute of Technology
Cambridge, MA 02139

Attention: Professor Arthur C. Smith

Subject: Electrical Engineer & Master's Thesis of Lt. James R. Seeley

I have reviewed the attached Thesis of Lt. James R. Seeley on behalf of The Charles Stark Draper Laboratory. The report is within the scope of the Thesis Proposal as previously approved and does not contain any material that is objectionable to The Charles Stark Draper Laboratory. It is also approved for its technical content.

It is understood that this Thesis will be the permanent property of MIT and will be placed in the MIT Library within one month after the date of submission. The report may not be published wholly or in part (except for such limited publication as contractual and internal company obligations may require) without the authorization of the Head of the Department of Electrical Engineering and Computer Science.

(Authorized Official of
The Charles Stark Draper Laboratory)

This thesis is dedicated to my wife Connie
and
to my son Matthew.

ACKNOWLEDGEMENTS

The author wishes to express sincere thanks to his two thesis supervisors, Dr. Albert L. Hopkins, Jr. of the Draper Laboratory, and Assistant Professor Hoo-min D. Toong, of the Massachusetts Institute of Technology, for their guidance, and their many useful suggestions and constructive criticisms.

An additional debt of gratitude is owed to the personnel of the Digital Development Group (10L) of the Draper Laboratory for providing a friendly and stimulating atmosphere in which this research was performed. In particular, special thanks go to Dr. T. Basil Smith, III, Charles Smith, and Alex Kemp for their numerous suggestions and improvements in this research effort.

This thesis was sponsored by National Science Foundation, Contract Number DCR74-24116, and by the Office of Naval Research, Mathematical and Information Sciences Division under Contract Number N00014-76-C-0502.

Finally, thanks are also due to Mary Shamlian and Linda Sciarretta for the typing of the final version of the thesis, and to the Draper Laboratory's Publication Department for assistance in the preparation of a great many of the figures.

DEVELOPMENT OF A HIERARCHICAL, DYNAMICALLY
RECONFIGURABLE INPUT/OUTPUT NETWORK

by

Lt. James R. Seeley

Submitted to the Department of Electrical Engineering on May 12, 1977, in partial fulfillment of the requirements for the Degrees of Electrical Engineer and Master of Science in Electrical Engineering.

ABSTRACT

This thesis traces the development of the hierarchical, dynamically reconfigurable, input/output network which has been constructed at the Draper Laboratory. It presents a summary of the design procedures used in determining the network architecture, communication methods, message formats, and overall topology. Further, it describes both the hardware and software features that have been implemented in the network's microprocessor-based nodes. In addition, the centrally located software algorithms developed to configure, repair, and monitor the network have been extensively discussed. Finally, the thesis also includes a reliability analysis of the network in a typical application, and a performance evaluation of the effectiveness of the configuration and control programs.

The implementation of a hierarchical, dynamically reconfigurable network is a radical departure from the typical data bus oriented I/O systems found in many applications today. The justification for this departure lies in the improved damage - and fault-tolerant features, not found in other architectures, that the network possesses. Specifically, through the alternate path redundancy provided by the inactive links, the centralized controlling element is capable of dynamically reconfiguring the surviving portions of a damaged network in order to circumvent the malfunctioning elements. Thus, the overall reliability of the I/O system has been improved, since it is now possible to maintain communication with each peripheral node and its host processor, in spite of the occurrence of moderate levels of physical damage.

Two variations of the basic network design have been developed. One, termed the single level network, is the standard form of the I/O net used in conjunction with the Laboratory's OSIRIS (Onboard Survivable Integrated Redundant Information System) demonstration implementation of a commercial aircraft flight control system. All nodes in the single level network are on the same hierarchical level and consequently communicate in an identical manner with the central computer. The second variation of the basic design is called the bilevel network. In this case, two separate hierarchical levels exist independently, joined at only one point of tangency, the bilevel node. The advantage of the bilevel network over the single level network arises in applications where the computational load is great at the various local processors. Since the bilevel network is able to effectively isolate the computationally intensive nodes in a lower level network, not in direct communication with the central processing element, an increased processor throughput is potentially possible.

To date, experimentation with a six-node single level network has indicated that the percentage of the available I/O bandwidth required for network management functions is compatible with the operation of the digital autopilot application program. Additionally, the average time required to detect a fault, load the software reconfiguration task, and correct the indicated malfunction, given the characteristics of the hardware currently implemented, is on the order of 1 sec. Finally, the software overhead in the central computer for the network control programs has amounted to less than one thousand sixteen bit words, plus an added three hundred words for system tables and constants. Overall, the hierarchical, dynamically reconfigurable I/O network, is conceptually well suited to the broad range of applications requiring a high availability input/output system.

Thesis Supervisor: Albert L. Hopkins, Jr.

Title: Staff Member, The Charles Stark Draper Laboratory, Inc.

Thesis Supervisor: Hoo-min D. Toong

Title: Assistant Professor of Electrical Engineering.

TABLE OF CONTENTS

Page

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The potential role of distributed processing has become increasingly important in the real time control and data management environments of many commercial, industrial, and military systems. Because of the declining costs of hardware, the placement of a significant amount of computational capability at a remote location has become feasible. However, the inherent physical separations, and proliferation of additional sub-systems resulting from such a distribution, have resulted in a dramatic increase in the volume of input/output (I/O) communications required. Clearly, it is essential that this added communications load must be made as reliable as possible in order to satisfy the stringent performance requirements imposed by most high-availability systems [24].

To achieve highly reliable inter-device communications, some type of fault-and damage-tolerant implementation is normally needed. Taking the form of redundant hardware, alternate communication paths, or error recovery procedures, for example, these features provide a distributed I/O communications system with the capability for "graceful degradation" [3] [16]. In other words, the system can continue to operate correctly, even in the presence of a predetermined threshold of hardware or software faults, or after incurring moderate levels of physical damage. Many such I/O systems exhibiting a wide range of fault-tolerant capabilities have been built during the past several years [24]. This thesis will trace the development of one such implementation for a commercial aircraft flight control system that has been constructed at the Charles Stark Draper Laboratory.

Since the late 1960's, the Draper Laboratory has been investigating the application of fault-tolerant multiprocessing technology to a variety of digital control applications, primarily to the area of aircraft flight control systems [16] [25] [29]. Out of this continuing investigation has emerged the OSIRIS (Onboard Survivable Integrated

Redundant Information System) concept. OSIRIS is a real-time distributed processing system consisting of one or more fault-tolerant multiprocessors, a damage-and fault-tolerant network, physically separated local processors, and operational software for fault detection, identification, and recovery [15] (see Figure 1.1). An experimental version of OSIRIS is presently in existence, and it is the OSIRIS input/output network which is to be studied in this thesis.

The OSIRIS fault-and damage-tolerant network, first envisioned in 1973 [29], was originally implemented in 1974 as a six-node simplex network of hardwired circuit switches under central software control [25]. This demonstration was evaluated under an Office of Naval Research contract to investigate the possible application of the network approach to shipboard data management systems. Soon after this original effort had been successfully completed, the rapid rise of low cost microprocessors signalled that a more flexible follow-on implementation could be realized by centering the design of the network node around a microprocessor. Consequently, work was initiated in early 1975 to incorporate the microprocessor into the demonstration network. At the same time, it was decided to utilize as the central processing center for the network the CARDS multiprocessor, another C.S. Draper research effort. With this decision the "breadboard" OSIRIS system had begun to take shape. The central processing center, or CPC, possessed an extensive assortment of fault-and damage-tolerant features. It was able to execute, simultaneously, multiple tasks in addition to the software required for the I/O network configuration and control. Some of the more significant features of the CARDS multiprocessor are as listed below:

1. triply modular redundant processors
2. triply modular redundant memories
3. triply modular redundant I/O modules
4. triply redundant serial I/O bus
5. dual redundant line drivers and receivers
6. additionally, each I/O module contained among other things, bus isolation gates and error detection and recording circuitry.

Subsequent to the beginning of work on the network of microprocessor nodes, a follow-on to the previously cited Office of Naval Research contract was received. The objective of these additional funds was to demonstrate a concept, proposed earlier in reference [26], known as a "bilevel network". This concept evolved as an attempt to more fully realize the hierarchical system potential of the general

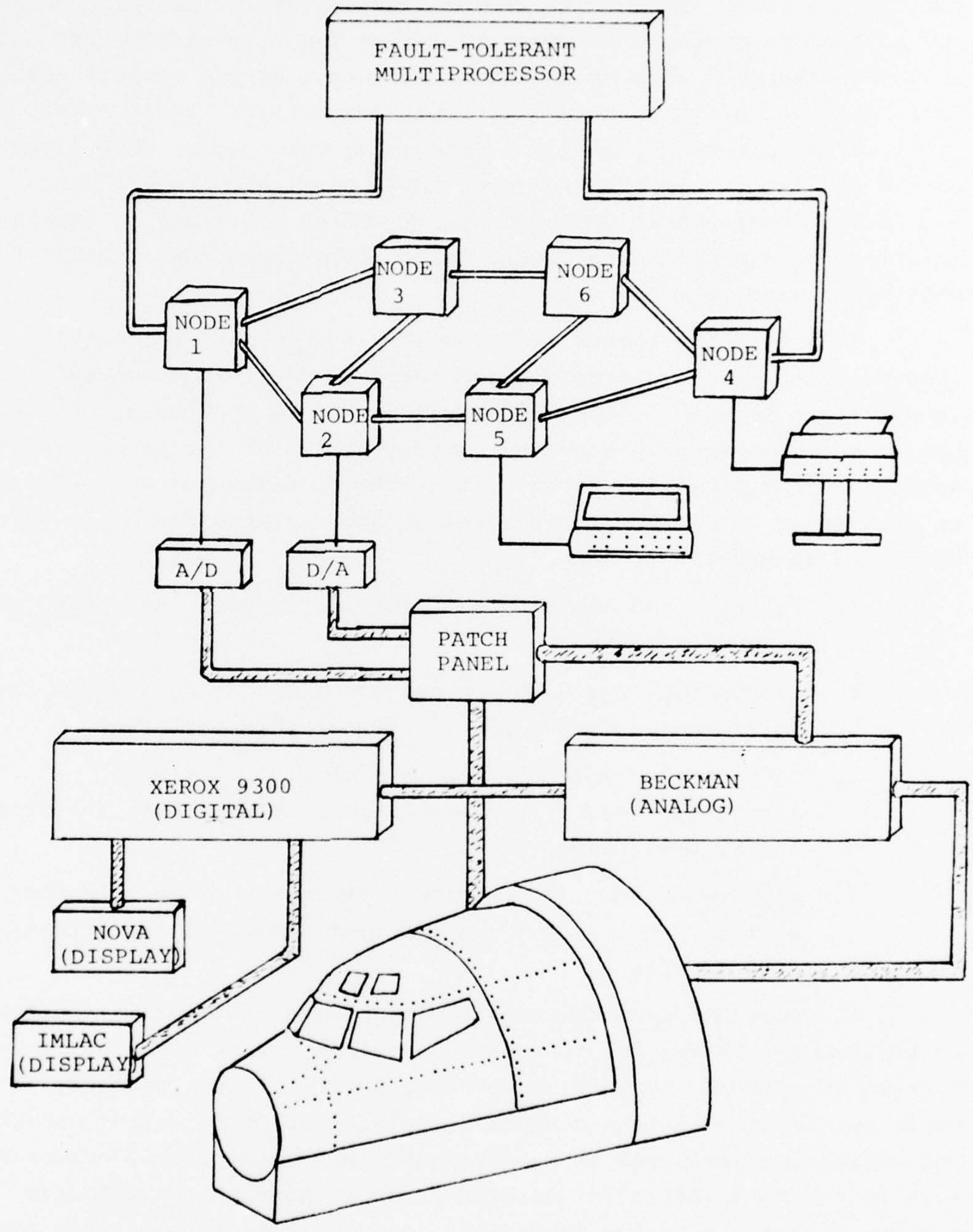


Figure 1·1 Block Diagram of Demonstration OSIRIS System.

network architecture [26]. In a bilevel network, certain specialized nodes, known as "bilevel nodes", would allow two distinct smaller networks to exist independently, joined only at a single point of tangency, the bilevel node. If the two networks were hierarchically related (i.e. one of them subordinate to the other) then the bilevel network could effect a clear division of the network control versus computational responsibilities. This characteristic, potentially, could result in a significant improvement in local processing throughput, when compared to the results obtainable from the simple or single level network already in development. Through fairly modest additions to the hardware for the microprocessor node, it was determined that a bilevel node could also be constructed.

Thus, as this thesis began, work was underway, not only on the incorporation of the microprocessor into the fault-and damage-tolerant input/output network of the demonstration OSIRIS system, but on the preliminary stages of the eventual modification of the single level network into a bilevel network. It is the objective of this effort to contribute to both of these research goals stated above, in the following manner:

1. To trace the design processes used in both the single and bilevel networks.
2. To describe the hardware and nodal operating systems that have been developed by C.S. Draper personnel [28].
3. To develop, implement, and evaluate the software algorithms needed to control and configure both the single and bilevel networks.
4. To present some indication of where the current network research effort should proceed next, based on the progress made to date.

Although the approach to a fault-and damage-tolerant I/O network as implemented in the OSIRIS system is unique, other existing computer systems do utilize, to varying degrees, fault-tolerant features in their inter-communication schemes. Besides the normal error detection and correction performed on incoming data by most systems, networks such as the ARPA (Advanced Research Projects Agency) network also possess a reconfiguration capability [6]. This feature is made possible through the use of its packet switching node computers known as IMP's (Interface Message Processors). In the absence of normal traffic,

each IMP transmits idle packets on unused lines at half second intervals. The lack of a return packet or incoming traffic indicates a faulty line and allows the dynamic routing tables at each node to be updated accordingly. Thus, faulty links are bypassed and previously spare links are activated automatically [23]. Though not as complex, CYBERNET, of the Control Data Corporation, also has a fault-detection and correction capability, but it requires human intervention to activate the redundant links [17]. Still other telecommunications networks such as MERIT at the University of Michigan and OCTOPUS at Lawrence Livermore Laboratories in California also implement redundant links to provide alternate communication paths in the event of link or processor failures [7]. In all the examples given, the need for a highly reliable intercommunications system is satisfied to a great extent by utilizing the dual concepts of redundancy and reconfigurability. These two features are also found to be dominant in the design of the OSIRIS input/output network.

In the chapters to follow, the development of both the single and bilevel networks is examined. Chapter 2 discusses the various decisions that were made during the preliminary stages of the network design. Chief among these include:

1. I/O architecture selection
2. Communications method selection
3. Link and data characteristics
4. Topology selection

Chapter 3 expands, in fairly great detail, upon the features of the single level six-node network. Included in its discussion are the following topics:

1. General description and capabilities
2. Demonstration topology selected
3. Nodal hardware
4. Nodal operating system
5. Network configuration and control software
6. Incorporation of network control and configuration software into the CPC

Chapter 4 repeats an identical development of the bilevel ten-node

network. Unfortunately, the complete hardware implementation of the ten-node network has not yet been completed. Chapter 5 describes the general problem of evaluating the reliability of a typical OSIRIS I/O network. It also discusses, assuming a number of apriori conditions, the increased reliability characteristics that the alternate paths in the six-node network contribute. Chapter 6 treats the subject of the performance of the single level network, particularly in the area of the configuration and control algorithms' evaluation. Finally Chapters 7 and 8 present topics for future investigations, and thesis conclusions, respectively.

CHAPTER 2

FAULT-TOLERANT NETWORK DESIGN ALTERNATIVES

2.1 Design Constraints Imposed by the Airborne Environment

The initial step in the development of any general communications network is to identify the set of parameters over which the design is to be a function. Among the more important of these items are the following [20]:

1. Total cost
2. System reaction time
3. Network survivability and vulnerability considerations
4. Network efficiency
5. Network user requirements
6. Serial or parallel transmission
7. Circuit-switched or packet-switched procedures
8. Message routing procedures
9. Network control
10. Security
11. Network topology

For the OSIRIS input/output network, several of the above design considerations, when placed in the context of the airborne environment, form a set of constraints over which the network design must be optimized. Specifically, the following are the important questions to be answered during the network design process:

1. Total Cost

   How can the desired levels of performance and reliability be achieved, while still minimizing the total cost, and often more importantly, the total weight of the system?

2. System Reaction Time

Can the functions of network configuration and control be accomplished in a relatively small percentage of the total I/O bandwidth? If not, their utilization will interfere with the primary flight control functions of the overall system, thereby degrading the system reaction time.

3. Network Survivability and Vulnerability Considerations

What is the desired level of reliability required? How is it defined? How much tolerance to faults and physical damage should be incorporated? How best should these survivability features be implemented?

4. Serial or Parallel Transmission

What method of communication is best suited to the network? How is it implemented? What do the actual data links look like? Is the communications method chosen compatible with the I/O bus of the CPC?

5. Circuit-Switched or Packet-Switched Procedures

How does the hierarchical architecture affect the selection of nodal switching procedures? How does transmission delay affect both methods? Is there an application where both alternatives could be utilized?

6. Network Control

What types of transmissions should be developed to control the network? What is necessary to effectively configure a network, verify its correct operation, and reconfigure it when a fault is detected?

7. Network Topology

Is there an optimum arrangement of data links for the network given the geographic locations of the nodes and the number of I/O ports per node? Over what characteristics is this optimization process to be conducted? How sensitive is the network performance to variations in topology?

Answers to this extensive list of important questions will be presented, not only in subsequent sections of this chapter, but also, to a varying extent, in all succeeding chapters. Throughout the development; however, a greater emphasis will be placed upon achieving the

required fault-and damage-tolerant capabilities, rather than upon minimizing such considerations as total cost, software simplicity, or the total weight of the linkage mass.

2.2 Selection of the Network Architecture Over Other I/O Alternatives

The decision to utilize a network scheme for the OSIRIS input/output communications method was based on a careful examination of the advantages and disadvantages of the various fundamental fault-tolerant communications architectures. This comparison, as summarized in Table 1, served to differentiate between the three most common structures; the dedicated or star connection, the redundant bus connection, and the network connection (see Figure 2.1). Though each alternative possesses definite strengths and weaknessess, the overriding concern in the OSIRIS application of providing an uninterrupted data stream to the central processing center, tended to narrow the spectrum of acceptable I/O alternatives. Specifically, the communications structure to be selected must, among other capabilities, be able to continue to function correctly in the presence of moderate levels of physical damage [15]. Additionally, it is desired that the occurance of isolated faults in one node have a minimal effect on other nodes. In other words,the faults must be uncorrelated in order that the validity of the data stream not be degraded [16]. With these restrictions and the comparisons of Table 1 in mind, the network architecture was chosen as the logical choice for the OSIRIS I/O system.

In essence, the selection of the network structure for the OSIRIS system was a compromise between the dedicated and redundant bus architectures. Like the redundant bus, a network has a large linkage mass resulting from the necessity of providing alternate communication paths. Yet, like the dedicated connection architecture, it does not require complete replication in order to achieve routing redundancy [25]. As an additional consideration,in a hierarchical design the network control intelligence is centrally located, often in a well-protected, redundant, highly damage-resistant environment. This is a key feature, for as long as the central processing element remains functional, the network as a whole is able to survive in the face of a partial loss of capability. Through its ability to reconfigure the remaining network connections, the CPC can isolate the damaged portion of the network and effectively circumvent it in most cases. Another noteworthy advantage of the hierarchical structure is found in the one-way initiation of all communications by the CPC. Since the central processing element is the top level of control in the system, all data, control, and configuration requests are originated in the CPC, thereby greatly simplifying the

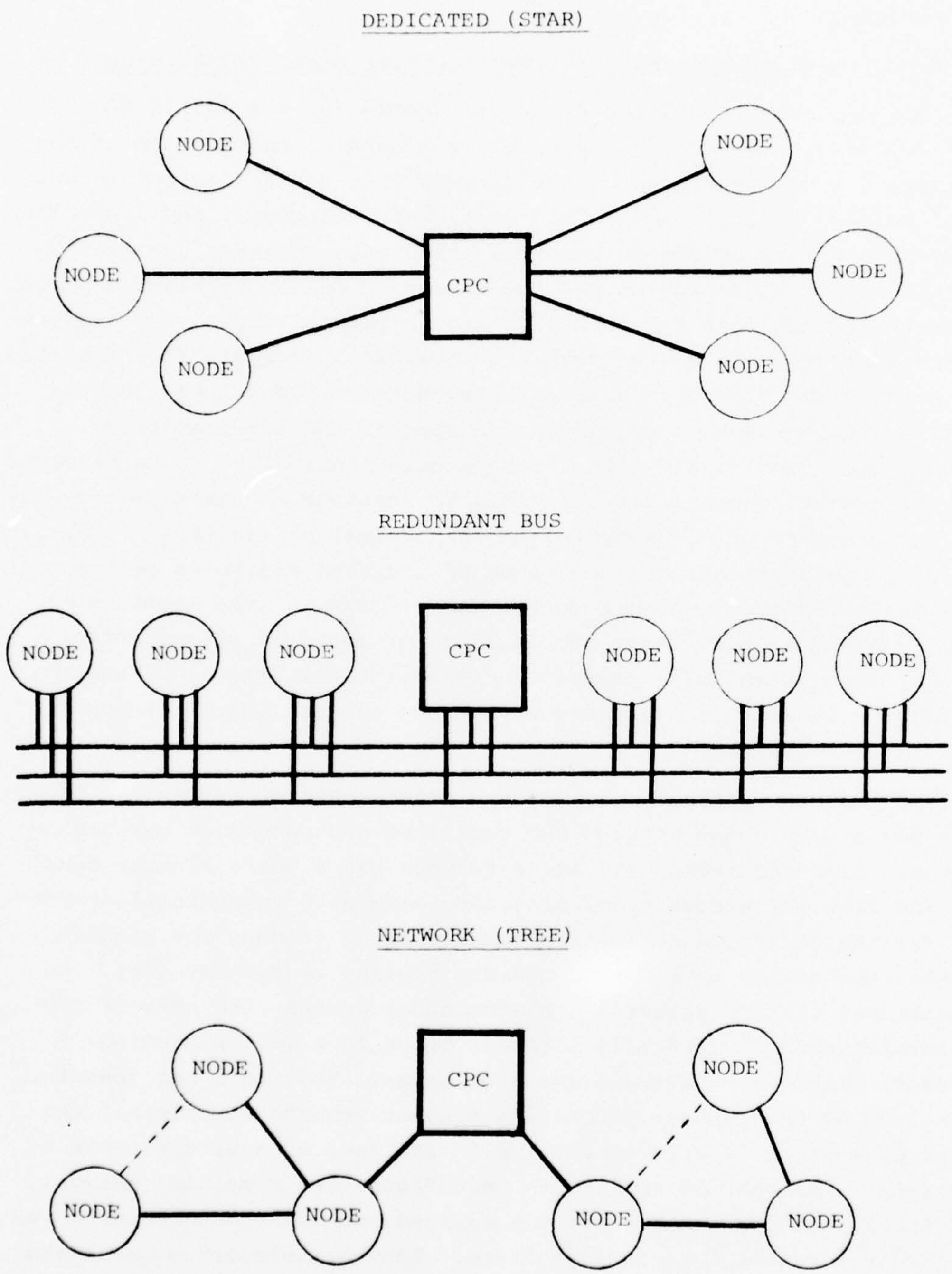


Figure 2.1 Data Communication Structures.

TABLE 1

COMPARISON OF MAJOR FAULT-TOLERANT
COMMUNICATION STRUCTURES

ADVANTAGES	DISADVANTAGES
1. Dedicated Connections (STAR Network)	
. Simplest to implement.	. Difficult to expand once central computer enclosure has been built.
. Failure of one node does not "bring-down" entire system (inherent "graceful degradation").	. Requires excessive wire weight and bulkhead penetrations.
	. Communication to node lost if dedicated link fails.
	. Underutilizes the connection medium and interface electronics.
2. Redundant Bus Connections	
. Fewer links and less cable weight than dedicated connections.	. More vulnerable to damage than network.
. Has good growth potential.	. Not readily adaptable to point-to-point fiber optic implementation (bus coupler design).
. Less complex than network (minimal operational and reconfiguration software overhead).	. Susceptible to "multiplier phenomenon"(single failure disabling a large portion of I/O system).
. Most widely used (less developmental risk).	
3. Network Connection	
. Greatest tolerance to physical damage (reconfiguration).	. Costly in associated node hardware and software overhead.
. Simplifies the implementation of hierarchical processing systems.	. More bulkhead penetrations and wire weight than nonredundant single bus system.
. Centrally located network control.	. Less generally accepted due to complexity of configuration control.
. Simplifies fault identification.	

message routing procedures [29]. Stated differently, no provisions need be made in the communications protocol to allow for node to node inter-communications, or to provide for transmissions to the CPC initiated by a node. In summary, although the redundant bus and dedicated connection structures are excellent I/O architectures for many classes of applications, the requirements put forth by the OSIRIS system dictate the use of a hierarchical network for its I/O scheme. In a related application, the network approach is also well-suited to a naval shipboard environment. In this case, many of the same restrictions aimed at insuring high levels of performance and reliability that apply in an airborne application, are also critical to the shipboard command and control function [17].

2.3 Serial or Parallel Transmission

Three considerations formed the basis for the selection of the transmission method for the OSIRIS I/O network.

1. Will the method be feasible in a widely distributed connection scheme involving several nodes and moderately long links?

2. How does the ease of implementation of the hardware communications interfaces compare for the two alternatives?

3. Is the method compatible with the internal I/O bus of the CARDS multiprocessor to which it will be attached?

On the basis of these three concerns, it was decided to use serial asynchronous transmission throughout the network. Specifically, this was the only one of the two alternatives which was both easy to implement, and compatible with the central processing center's I/O bus. In addition, opting for serial instead of parallel communications interfaces reduced the total linkage mass requirements by a ratio of approximately eight to one, a significant savings in cost for any network.

In the breadboard OSIRIS system standard microprocessor asynchronous interface adapters (ACIA's) were chosen as the hardware element to be the transmission controllers [26]. The standard RS-232 [18] 60 mA current loop was selected as the basis on which the ACIA's would transmit a non-return-to-zero (NRZ) type code. Although asynchronous operation does require additional start, stop and parity bits to be sent with each byte transmitted, the resulting reduction in the

effective throughput is not significant in the current implementation. Still, for any follow-on version of the OSIRIS system, a much greater I/O bandwidth will be required. This need will be satisfied, most likely, not by converting to a synchronous communications scheme, but by operating the ACIA's with faster microprocessors and associated memories. In this way, the problems involved in a byte synchronous system of transmitting and interpreting correctly the sync signals, can be avoided [23].

As far as the compatibility of the network transmission method with the CPC's I/O bus was concerned, the decision to utilize serial asynchronous transmission was predetermined by the existing bus architecture. The internal bus of the CARDS multiprocessor is a triply redundant serial I/O bus. To interface it with the network, devices known as I/O access units (IOA's) have been constructed [16] (see Figure 2.2). The IOA's contain a voting mechanism to convert the triply redundant data of the internal bus into a single majority signal [28]. The resulting signal is then routed to an ACIA where the requisite start, stop, and parity bits are appended before transmission to the network. The process is reversed for incoming data from a particular node. A copy of the returning signal is placed on each I/O bus so that it can be distributed to the applicable ACIA's in the CPC.

In conclusion, the decision to implement serial asynchronous transmission was made primarily because of the structure of the CPC's triply redundant I/O bus, the ease of implementation using readily available ACIA's, and the great savings in wire weight when compared to that necessary in a parallel system. Furthermore, the I/O bandwidth required by the OSIRIS flight control system was not high enough to warrant the added expense of either byte synchronous or parallel communications.

2.4 Circuit-Switched or Packet-Switched Procedures

An important determination in the overall design of the OSIRIS I/O system was whether circuit-switching or packet-switching procedures should be utilized. The characteristics of both switching methods are delineated in Table 2. In essence, circuit-switching involves two operations. First, a communications path must be established between the sender and receiver. Once established, the information transfer is then allowed to take place. On the other hand, packet switching is one example of a common technique known as "store-and-

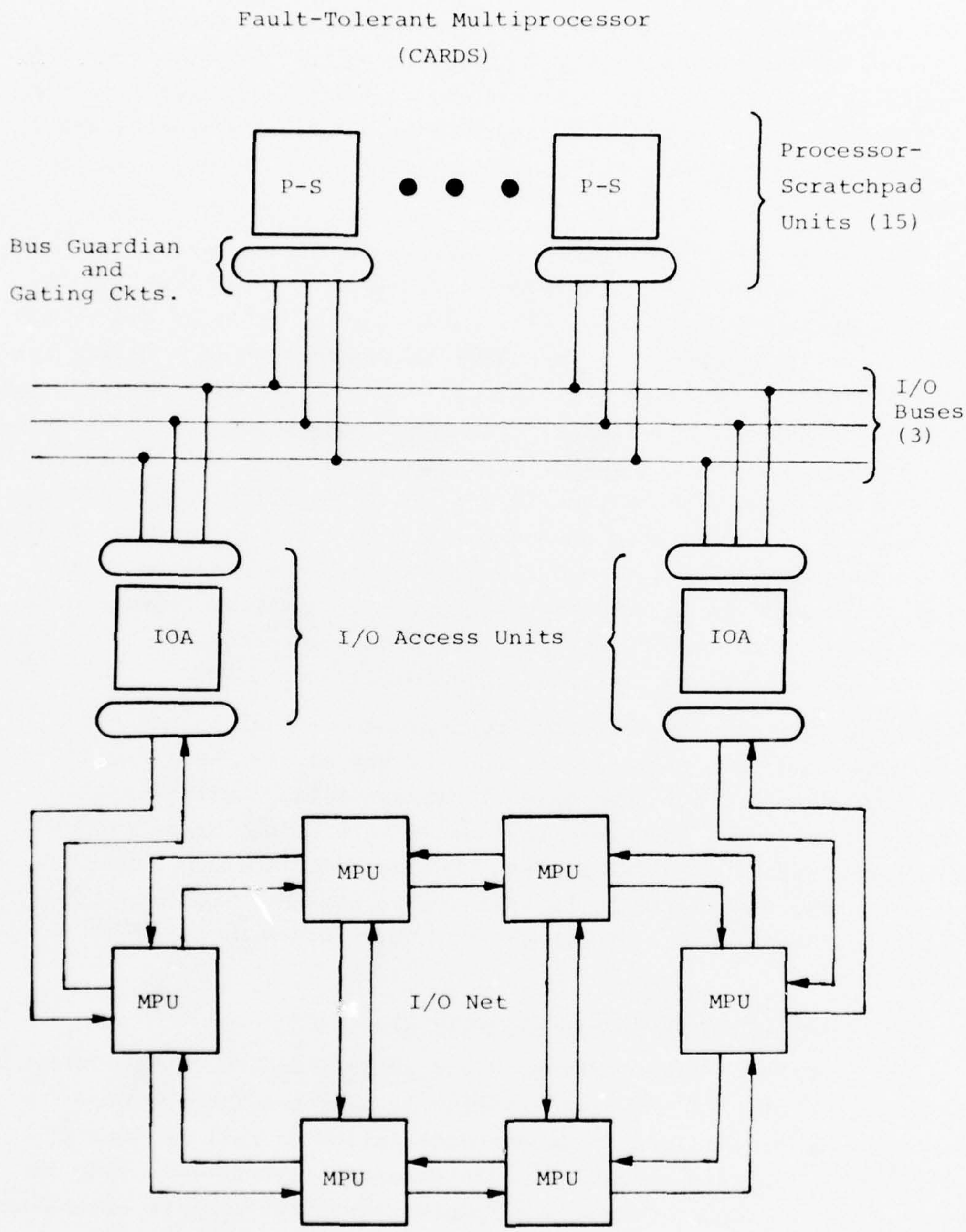


Figure 2.2 OSIRIS I/O Bus Block Diagram.

forward" [11]. In this method a message is stored at intermediate nodes as it makes its way toward its destination. Each time the message is forwarded correctly, the previous node is freed from any further responsibility for the message upon receipt of a positive acknowledgement. Additionally, in a store-and-forward system some sort of routing strategy is required at the node itself, in order to select the next intermediate node to which the message will be forwarded. In a full scale packet-switching network, the ARPA net [6] for example, the bandwidth of the various channels is more effectively utilized since the routing strategy can be a dynamic function of such parameters as the I/O traffic load [11].

Returning to the OSIRIS system, the decision as to which switching scheme to implement in the microprocessor nodes, as has been done throughout the design process, was based upon the intended application environment of the network. For the single-level I/O network, circuit-switching was selected due to its relative simplicity of implementation, and its characteristic of negligible transmission delays. This is attributable to the fact that once a network is configured, it is essentially a bus structure over which messages are sent and received, with no intermediate processing by nodes in the transmission path. For the bilevel network, however, a combination of circuit-switching and packet-switching was neccessary. This stemmed from the requirement to be able to communicate between the two levels of the hierarchy. To send a message from the central processing center to a node in the lower level of the network, the desired transmission is circuit-switched through the upper level network to a particular bilevel node. Since the original message is directed specifically to that bilevel node, the node stores the entire message in its input buffer. Upon interpretation of the message, the bilevel node decides that data is required from one of the nodes subordinate to it in the lower level. The bilevel node then reformats the message and forwards it to the actual destination node. The subsequent returning data is routed back to the CPC in a reverse manner. The bilevel node handles the packet-switching procedures identically to a larger scale network's node except for four basic differences:

1. It has no dynamic routing capacity
2. It cannot queue messages
3. Its buffer lengths are relatively limited

TABLE 2

COMPARISON OF CIRCUIT-SWITCHING AND PACKET-SWITCHING

MAIN CHARACTERISTICS OF CIRCUIT-SWITCHED SYSTEMS	MAIN CHARACTERISTICS OF PACKET-SWITCHED SYSTEMS
1. Logical equivalent of a wire circuit connecting the source and destination.	1. No direct connection established.
2. Real time capability, negligible transmission delay.	2. Real time capabilities limited by inherent retransmission delays.
3. Messages are not buffered.	3. Messages are buffered.
4. Hardware switches with minimal intelligence required.	4. Hardware switches with moderate switching computer required.
5. Message routing established prior to transmission.	5. Dynamic routing possible; however, some packets can become lost during message routing.
6. Any length transmission permitted.	6. Lengthy transmissions are chopped into short packets.
7. Does not allow for transmission rate or code conversion.	7. Buffering allows for speed or code conversion.
8. Fixed bandwidth transmission.	8. Variable bandwidth according to need.
9. Explicit messages.	9. Delegation of authority possible.

4. It transmits and receives packets of only one byte in length.

The implementation of packet-switching procedures in a network operating in a real-time control environment has one serious drawback. As stated earlier, this limitation is the inherent transmission delay associated with the packet-switching process. Since an iterative control loop program, such as the OSIRIS digital autopilot, cannot function effectively when significant delay is encountered in the transmission path, the bilevel network, as currently being designed, may not be practical due to the placement of the autopilot function solely in the CPC and the use of packet-switching in the bilevel nodes. Fortunately, this restriction can be easily eliminated, if the autopilot task is distributed amongst the bilevel nodes. In this way, each local processor will be in direct circuit-switched communication with its controlling superior, and consequently will not encounter a transmission delay.

2.5 Network Control Considerations

To effectively control the operation of the OSIRIS I/O network, a set of network "commands" consisting of easily implemented, reliable formats must be established. Through these short message transmissions, the network configuration and control algorithms residing in the CPC, and described in Chapter 3 and 4, can carry out the following four necessary control functions [5]:

1. Link creation and deletion.
2. Connectivity monitoring.
3. Reconfiguration.
4. Verification of the status of active and spare assets.

The following outlines the network commands currently in use in the breadboard OSIRIS I/O network [28]:

1. GATEMAN Command

   The GATEMAN Command is used in control functions (1) and (3) to set a Particular I/O port to the INBOARD state. An INBOARD port is a port over which messages may be received from the CPC. Once received, a message is routed through the node's internal switching circuitry (described in Section 3.3) and out to other nodes via any OUTBOARD ports. No message may be routed through a node unless an INBOARD port has been established on that node. Only one INBOARD port is allowed per node. Furthermore, the joining of a particular node to the I/O network is

signified by its acceptance of an INBOARD port. A GATEMAN command is three bytes long, two bytes of which are the destination node ID and its complement. To be interpreted properly by the nodal operating system, both bit strings must match exactly those stored in the destination node. This feature decreases the probability of erroneous message transmissions being able to alter a port to the INBOARD state.

2. CONTROL Command or RECONFIGURATION Command

The CONTROL command is also used in control functions (1) and (3), primarily to set a particular I/O port to the OUTBOARD or NULL state. An OUTBOARD port, as previously discussed, serves as the outlet for messages to flow from one node to another node, while a NULL port allows no message routing to to be performed by that port. Both commands have similar formats to the extent that they both are four bytes in length and contain the destination node ID and redundancy word as in the GATEMAN command.

3. RESTART Command

The RESTART command is a variation of the GATEMAN command in that it may be received and processed by a node which does not have an INBOARD port. CONTROL commands, incidentally, are not processed by a node unless the node has an INBOARD port. The RESTART command is essentiai to control functions (1) and (3).It's objective is set to the NULL state all ports on a particular node, and to reset certain software functions as will be described in Section 3.4. The real value of the RESTART command is seen in its use with the RECONFIGURATION function (3). In that case, if a node is detected as sporadically transmitting erroneous data over its one or more OUTBOARD ports [i.e. a "babbling" node [29]], the RESTART command can be utilized to disable this node, if possible, by NULLing all of its I/O ports. Thus, the effects of the malfunctioning node can be eliminated. The ability to silence a babbing node with the RESTART command is a direct benefit also of the implementation of duplex links throughout the network (refer to Section 3.3).

4. STATUS Request

The STATUS request is a particular type of CONTROL command used in control functions (1) (2) (3) (4). Its function

is to interrogate a node as to the state of its I/O ports. The node's response is then transmitted, after a slight delay, back to the CPC. If for some reason the node is inoperative or possibly no longer connected, the CPC will "timeout" while waiting in a loop for the expected response. In this case, an error condition is signalled. The number and duration of the CPC timeouts is an important factor in the performance of the various control and configuration algorithms (refer to Chapter 6). The STATUS request is used, not only as a connectivity monitoring tool, but as a means of insuring the successful execution of the GATEMAN and CONTROL commands during the link creation and deletion function.

Two other message formats, though not involved with the network control functions, are nonetheless worthy of note since they provide the network with a data acquisition capability.

1. DATA message

The purpose of a DATA message is to transfer data between the CPC and the node application programs. The DATA messages use the synchronization scheme to be described in Section 3.3. They can be of any length, and are terminated by a special form of the last message to be covered, the ACKNOWLEDGEMENT word.

2. ACKNOWLEDGEMENT Word

The ACKNOWLEDGEMENT word (ACK) is used solely with DATA messages. It is designed to acknowledge the receipt of each word of a DATA message, to control the flow of data (refer again to Section 3.3), and as an "END-OF-MESSAGE" indicator.

2.6 Topological Optimization Considerations

Fundamental to the design of any fault-tolerant network is its ability to sustain a given number of element failures without a serious loss of performance. This attribute of "graceful degradation" is dependent, not only upon the reliabilities and availabilities of the individual system components, but also on the "topology" of the network. Topology here means the connection pattern of nodes and links given that the geographic locations of the nodes have previously been determined [6]. When this pattern is optimized with respect to given reliability, delay, and cost constraints the procedure is termed

"topological optimization" [9].

The process of topological optimization, while strongly influenced by the theorems of graph theory, is generally implemented for sufficiently large networks in a heuristic manner. This fact can be seen by considering the number of distinct topologies requiring evaluation for a given set of n nodes and m links. This number is given by the expression [9]:

$$\frac{(n(n-1)/2)!}{(n(n-1)/2-m)!\ m!}$$

Evaluating this expression even for a small network, say of six nodes and ten links results in a number of possible combinations of 15!/10!5! or 3003. While an analysis is possible for networks up to approximately 25 nodes, the final topology is further complicated by the additional "real-world" constraints of non-uniform traffic patterns, anomalous line costs, etc [19]. Consequently, for the OSIRIS I/O network, [since its final implementation will consist of considerably more nodes than are present in the breadboard model [16]], a recursive, non-analytical, evaluation procedure will be used to determine the most acceptable network topology. In essence, the topological optimization will consist of maximizing the number of node failures sustainable in any portion of the network before communications are lost, subject to the dual constraints of cost (i.e. - number of links and total sum of link lengths) and transmission delay.

The network topology evaluation will be based on the following characteristics:

1. The geographic locations of the nodes will be coincident with the aircraft sensors and effectors (see Figure 2.3). For the breadboard implementation, however, no replication of sensors/effectors will be allowed, nor will replication of the number of nodes servicing each sensor be evaluated. Also, since no hypothetical distribution nodes such as given in Figure 2.3 has been formulated, the six demonstration nodes will be arranged symmetrically to facilitate the display and evaluation process.

2. The network must be able to sustain at least two link failures before any node of the network is isolated from the CPC. This translates to the requirement

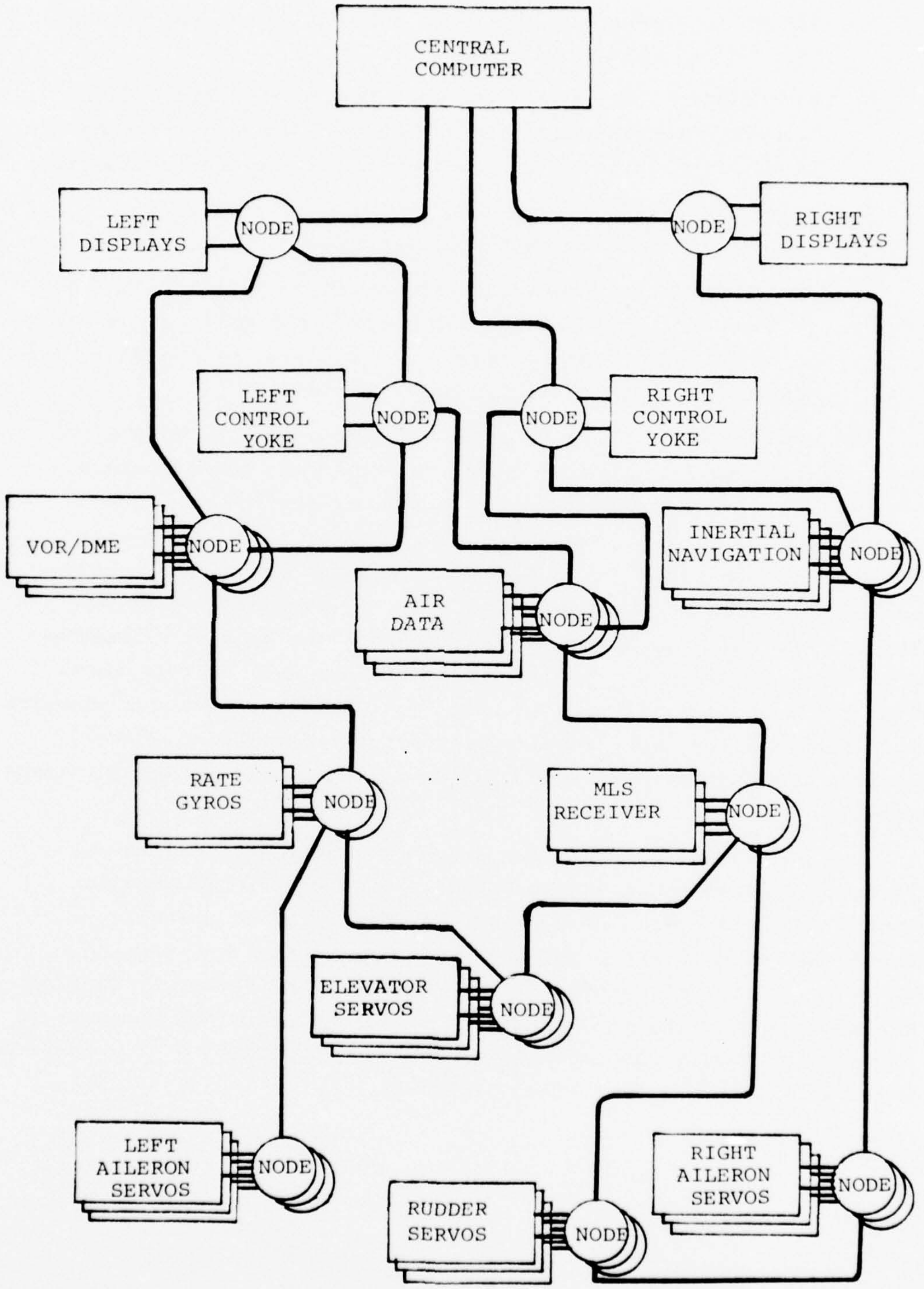


Figure 2.3 Hypothetical Distribution of Aircraft Node Locations.

that there must be three or more paths, though not totally disjoint, between each pair of network nodes, and between the CPC and each node.

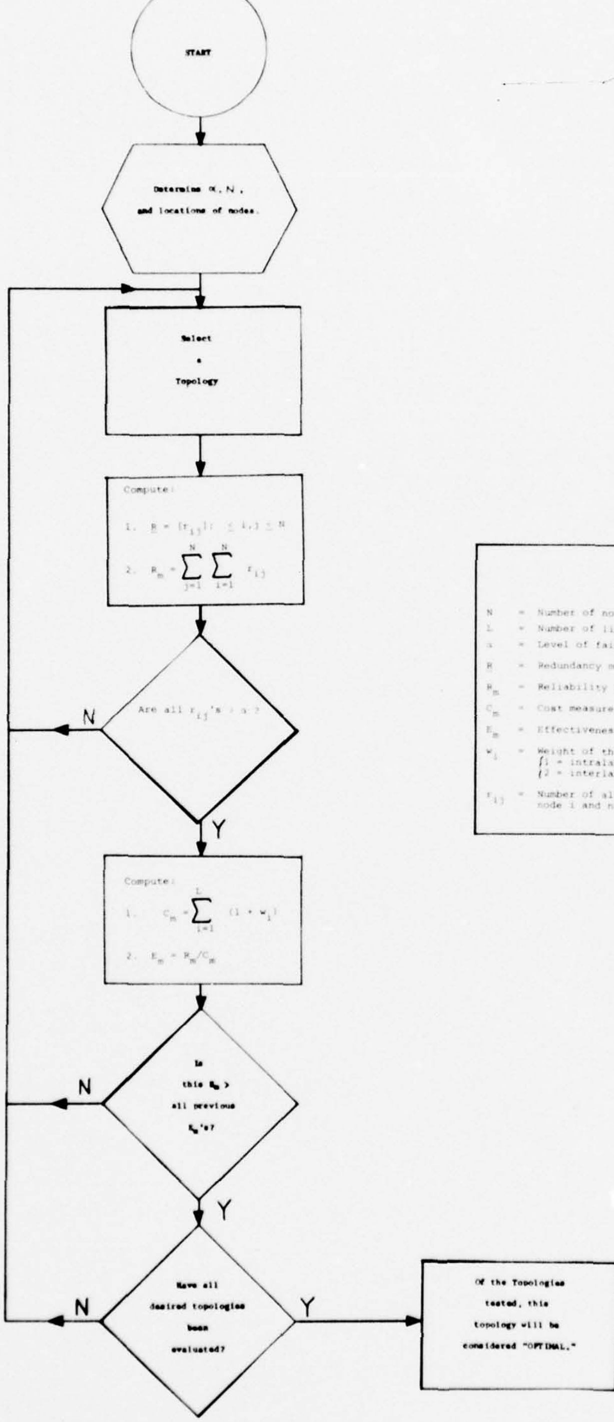
3. Each single level node will have three I/O ports. Each bilevel node will have six I/O ports. There will be no bilevel ports included in the initial topological evaluation.

4. Every node will be considered equal in priority (i.e. - all devices attached to each node are assumed to be equivalent).

5. Since no bilevel nodes will be included in the initial evaluation, the optimization process over the delay constraint will be eliminated. This occurs since there is negligible transmission delay in a circuit-switched network.

The topological evaluation procedure for a typical OSIRIS I/O network now evolves into constructing a minimum cost network given n geographically positioned nodes, such that a matrix known as the redundancy matrix $\underline{R}$ [6] is maximized with respect to a cost measure, Cm (the sum of the number of links and a weighted sum of the links). As seen in Figure 2.4, the evaluation cycle is repeated for as many different topologies as desired. The one with the highest effectiveness measure, Em, when the process is terminated will be considered "optimal" in the sense of this evaluation procedure. If two topologies or more have equal effectiveness measure at the conclusion of the evaluation process, then a random choice is made from the equivalent topologies.

In conclusion, although the stated evaluation procedure cannot be directly applied to an actual OSIRIS I/O network implementation without a number of broad assumptions, it does provide a simple, heuristic approach to the problem of selecting a test topology. On a large scale (i.e. - many nodes in the network), the number of topologies requiring evaluation before a decision could be determined would prove to be computationally unmanageable without the application of a computer implementation for the evaluation algorithm.



GLOSSARY OF TERMS

N = Number of nodes
L = Number of links
α = Level of failures to be tolerated
$\underline{R}$ = Redundancy matrix
R_m = Reliability measure
C_m = Cost measure
E_m = Effectiveness measure
w_i = Weight of the ith link {1 = intralab link; 2 = interlab link}
r_{ij} = Number of alternate paths between node i and node j

Figure 2.4 Flowchart of the Network Topology Evaluation Procedure

CHAPTER 3

SINGLE LEVEL SIX-NODE NETWORK

3.1 General Description and Capabilities

The six-node single level network is a damage-and fault-tolerant network implemented in the Advanced Digital Systems Laboratory of the Charles Stark Draper Laboratory. It consists of six Motorola M6800 microprocessor-based nodes, ten full duplex links terminated with differential line drivers and optical isolators, a display keyboard (DSKY) interface, a teletype interface, and an A/D and D/A interface (see Figure 3.1). The network is connected to the CARDS multiprocessor complex which functions as the central processing center (CPC), and superior "node" of the network. The primary function of the network, currently, is to serve as the I/O interface joining the CPC to a Boeing 707 flight simulator. The CPC executes an autopilot function, generating flight control signals based on simulated aircraft data generated by the Hybrid Simulation Facility (refer again to Figure 1.1).

The primary capabilities of the six-node network are listed as follows:

1. Maximum bandwidth of any link - 31.25 kHz.
2. Message format - circuit-switched packets of 11 bits (8 data bits, start bit, stop bit, parity bit), RS-232 standard .
3. Hierarchical command structure - no initiation of commands from nodes, and no communication allowed between two nodes.
4. Connectivity - three I/O ports per node.
5. Fault-tolerance level - network can withstand at least two link failures anywhere (except for the two links from the CPC) and continue to function as a fully connected network.
6. Communications method - asynchronous bit serial.
7. Memory capacities at the nodes - 1000 words of PROM, 256 words of RAM.

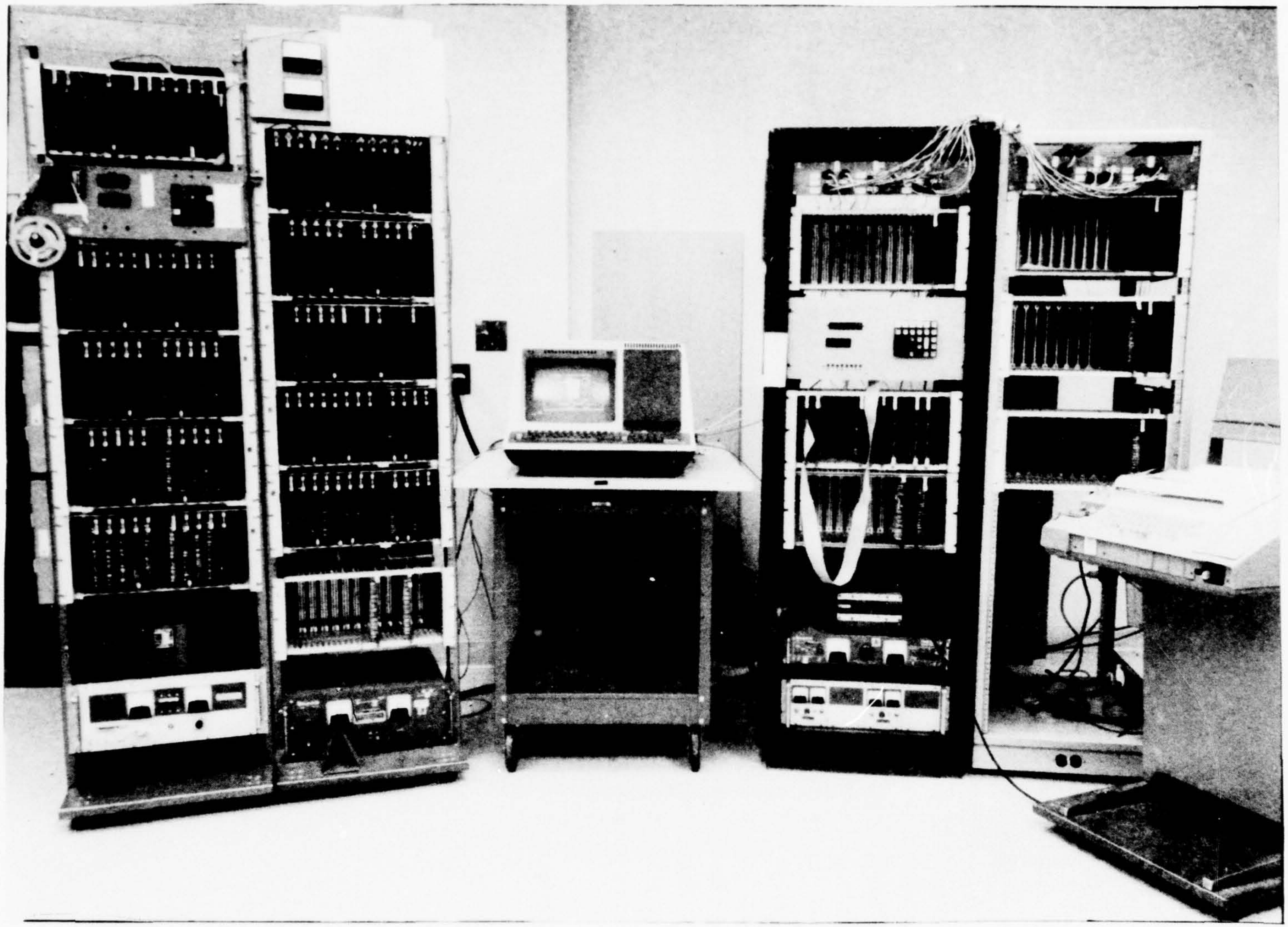


Figure 3.1 The Single Level Six-Node Network and CARDS Multiprocessor.

8. Error control - parity, framing, and receiver overrun checked on each byte. Error recovery initiated after receipt of one of above transmission errors, initiated by the CPC only.

9. Network control - software, located in the CPC's main memory, controls growth, reconfiguration and testing of the network.

10. Display - a Digital Equipment Corporation "DECSCOPE" shows the status of network at all times and allows for the interactive network monitoring program "SYSPROG" to be executed.

11. Modularity - each node is identical and can be placed in any node position. (i.e. this is because the node ID is "hard-wired" into the backplane of each node's slot.)

An expanded description of many of the listed network features will be presented in succeeding sections of this chapter.

3.2 Test Network Topology Selection

The topology of the six-node network was chosen on the basis of the topological optimization process as described in Section 2.6 for a network of six symmetrically positioned nodes. It was also selected so that it could be displayed clearly on the DECSCOPE, thus enhancing the demonstrateability of the single level network's features. The result is seen in Figure 3.2, a photograph of the actual DECSCOPE I/O network display. Few of the 3003 possible network combinations, as were calculated in Section 2.6, were actually evaluated, since many of the combinations were either equivalent, or else not applicable. For example, the cases of more than one link between a single pair of nodes were disregarded. For a selected topology, the redundancy matrix was calculated by determining the set of possible paths joining any two nodes. It should be noted that in the form of the matrix computed here, many paths contained one or more common links. Therefore, the failure of a particular link may reduce the redundancy matrix entry for a particular node pair by more than one. However, the redundancy matrix is still a valid interpretation of the level of fault-tolerance of the network due to the manner in which the network is actually reconfigured after a failure. It is also an interpretation

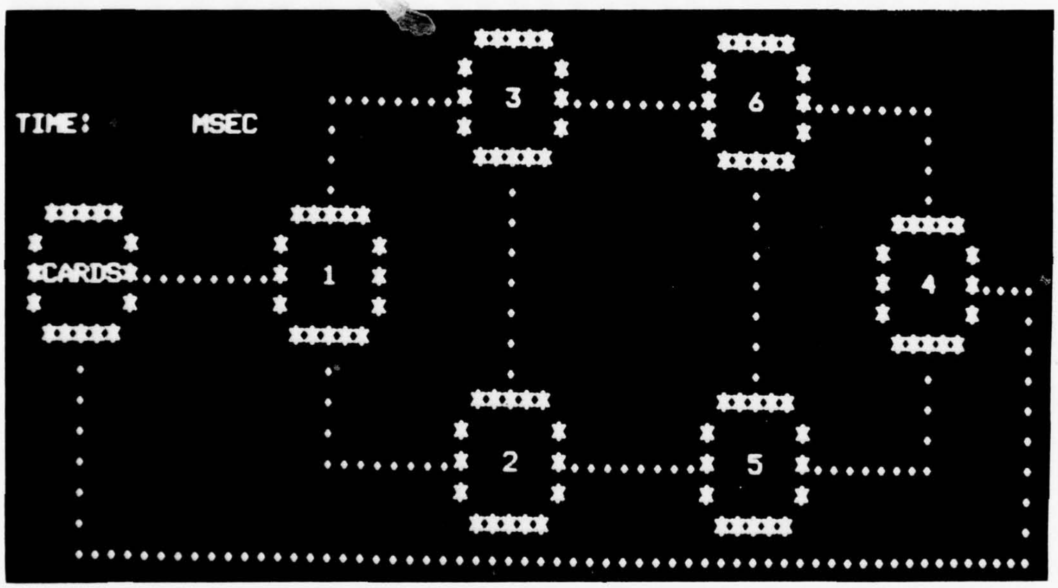


Figure 3.2 Single Level Six-Node Topology.

to be used in the discussion of the reliability test configurations (refer to Section 5.3).

Highlights of the optimization calculations used to evaluate the topology that was finally selected are as follows: (refer to Figure 2.4 for the optimization algorithm)

1. Redundancy Matrix

a.

$$\underline{R} = \begin{bmatrix} r_{11} & \cdots & r_{61} \\ r_{12} & \cdots & r_{62} \\ \vdots & & \vdots \\ r_{16} & & r_{66} \end{bmatrix}$$

b. Where r_{ij} is the number of alternate paths between node i and node j.

c.

$$\underline{R} = \begin{bmatrix} - & 4 & 4 & 8 & 5 & 4 \\ 4 & - & 4 & 5 & 4 & 4 \\ 4 & 4 & - & 5 & 4 & 4 \\ 8 & 5 & 5 & - & 4 & 4 \\ 5 & 4 & 4 & 4 & - & 4 \\ 4 & 4 & 4 & 4 & 4 & - \end{bmatrix}$$

2. Reliability Measure

$$R_m = \sum_{i,j} r_{ij} = 134$$

3. Cost Measure

a. $$C_m = \sum_{i} (1 + \omega_i) = 23$$

b. Where i = # of links and

$$\omega_i = \begin{matrix} 1 - \text{intra Lab link} \\ 2 - \text{intra Lab link} \end{matrix}$$

4. Effectiveness Measure

$$E_m = R_m / C_m = 5.83$$

Although the actual value of 5.83 for the effectiveness measure has no significance by itself, it did prove to be the highest value calculated for the networks tested. Also, as implied by the redundancy matrix, the fault-tolerance level of two link failures was met by this topology. In summary, the test topology of Figure 3.2 was chosen since it best satisfied the topological evaluation procedure by maximizing the redundancy of the network, while minimizing its cost of implementation.

3.3 Node Hardware Description

The M6800 microprocessor node is an efficiently designed digital system which is contained entirely on one plug-in circuit board. Its compact implementation is thus ideally suited to use aboard an aircraft or a ship. The node requires two power supplies for its TTL and MOS elements: +5 and -12 volts. It is partitioned into a control and data portion, in keeping with the current digital design philosophy.

Since the control section's finite state machine (FSM) is implemented primarily in software, its discussion will be deferred to Section 3.4 on the nodal operating system. From a strictly component sense, however, the control portion of the node's implementation is comprised of the M6800 microprocessor unit (MPU), 4 - National Semiconductor MM5204Q 512x8 bit programmable read only memories, two M6810 128x8 bit random access memories, miscellaneous TTL gates, etc., required to operate the microprocessor properly, and a 4 MHz crystal oscillator clock (see Figure 3.3). For a detailed description of the control signals of the M6800 MPU, and the other associated chips consult reference 26. The execution speed of the control portion of the node is hardware constrained. Suffice it to say that, although the M6800 MPU can operate at rates up to 1 MHz, it is currently operated at 500 kHz with the I/O in the divide by 16 mode (31.25 kHz). This is due to the relatively slow access times of the 5204 MOS PROM's. In an actual OSIRIS implementation, due to the use of faster microprocessors and bipolar fusible-link PROM's, the anticipated I/O speeds will be in the vicinity of 1 M baud.

The data section of the single level node has the important responsibilities for node I/O, and for circuit-switching the three ports onto and off of the node's internal bus. The data section also contains the hardware necessary to implement the physical links between

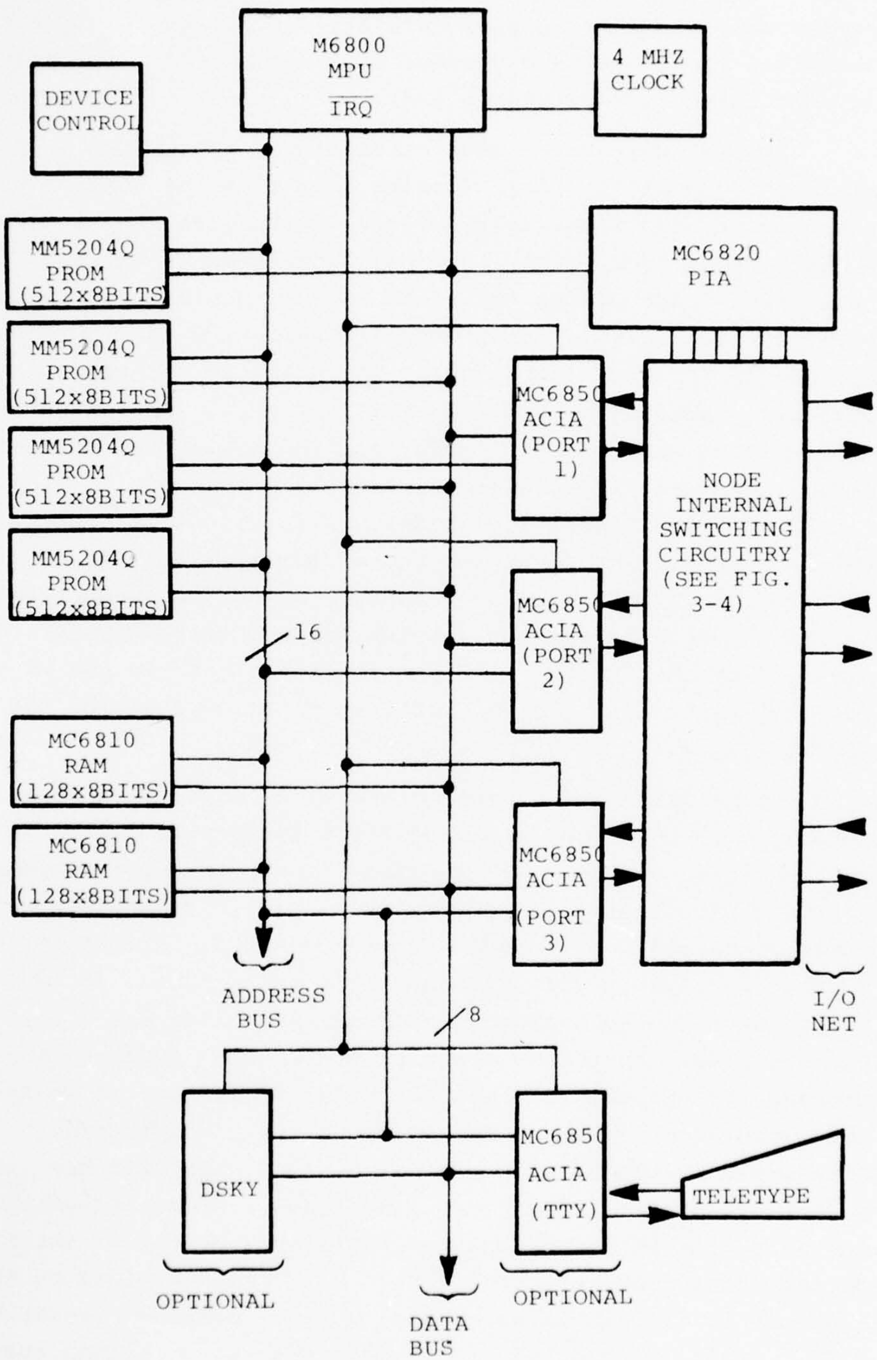


Figure 3.3 M6800 Microprocessor Node Block Diagram.

the nodes. The major components of the data portion are 4 - MC 6850 asynchronous communications interface adapters (ACIA's), two MC 6820 peripheral interface adapters(PIA's), 3 differential line drivers, 3 optical isolators, and various NAND gates, etc., to serve as the circuit switches (see Figure 3.4).

As far as the implementation of the data links is concerned, since the details of communicating data from the central processor to the network were adequately covered in the last chapter, only the actual composition of the physical links need be covered in this paragraph. Each of the ten links is full-duplex, and is implemented using 2 twisted, shielded wire pairs which are configured as two current loops (see Figure 3.5). To reduce noise, the links are optically isolated and differentially driven. Specifically, at the transmitting end of a particular half of a link, a differential line driver generates a differential signal in response to a direct current "1" or "0" (see Figure 3.5 again). This signal is transmitted in the current loop to the receiving end of the link. There an optical isolator is caused to either conduct, or not to conduct in response to it being either forward or reverse biased, respectively. The current produced by the optical isolator is then passed to one of the three ports of a particular node (refer to Figure 3.3 again) for appropriate message processing.

Once a byte of a message has been received by an ACIA an "IRQ" interrupt is generated (to be defined in Section 3.4). In response, the operating system will determine first, if the message is intended for that particular node's attention, and if so whether any parity, framing, or receiver overrun error conditions are indicated. If the message is not for that particular node, it is ignored. Since the intended path to the destination node was constructed during the configuration phase in a circuit-switched network, the message will proceed through the intermediate node, regardless of whether it is being processed or not. The route of its passage through the node is determined by the setting of the nodes circuit switches (refer to Figure 3.4 again). The circuit switches are enabled using specific bits of one of the two registers, of one of the PIA's found on the node. The setting of these bits is determined by the MPU, in response to CPC generated network control commands (refer to Section 2.5). Again, the restriction exists that no more than one I/O port may be set to the INBOARD state. This restriction prevents the

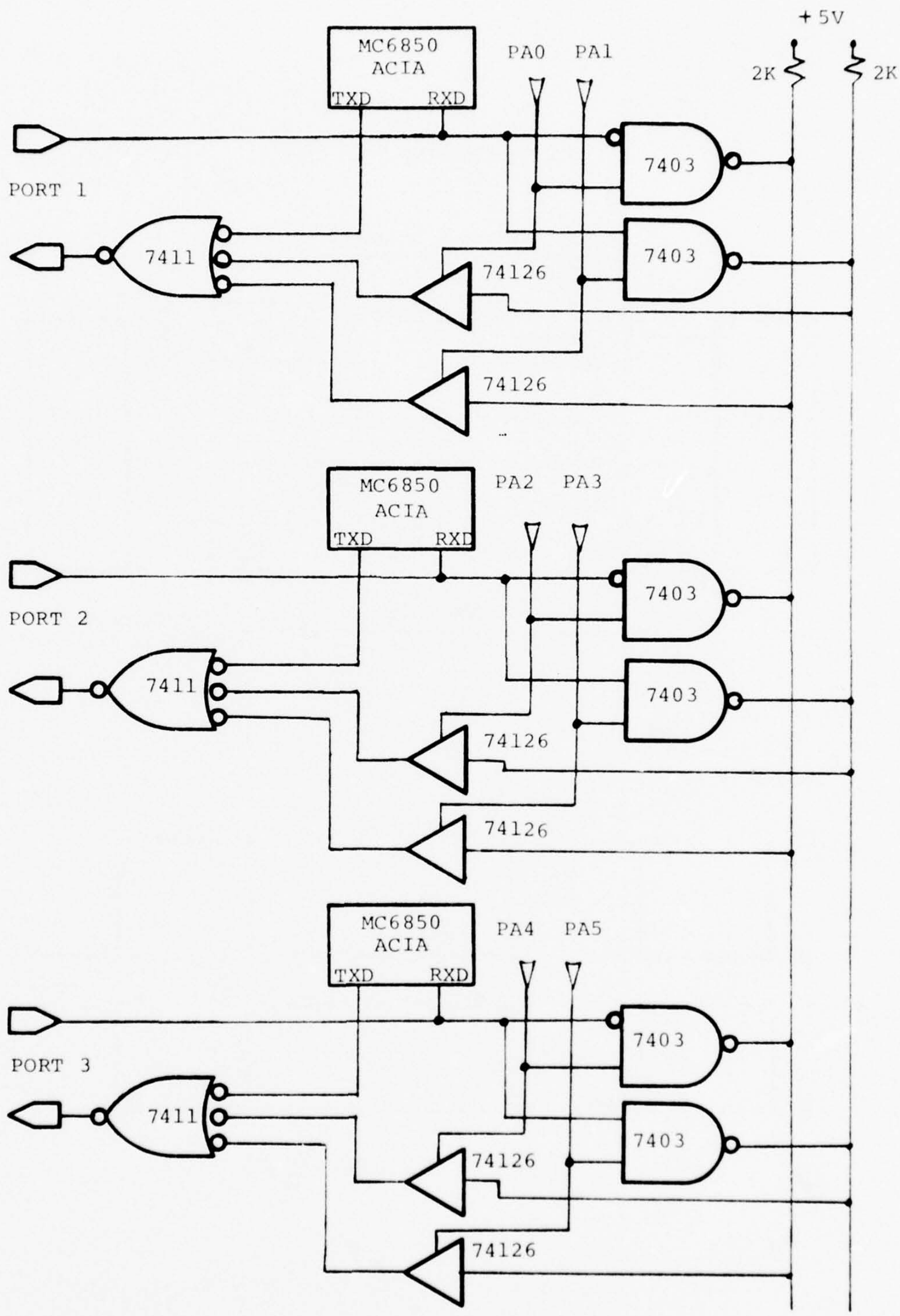


Figure 3.4 Node Internal Switching Circuitry.

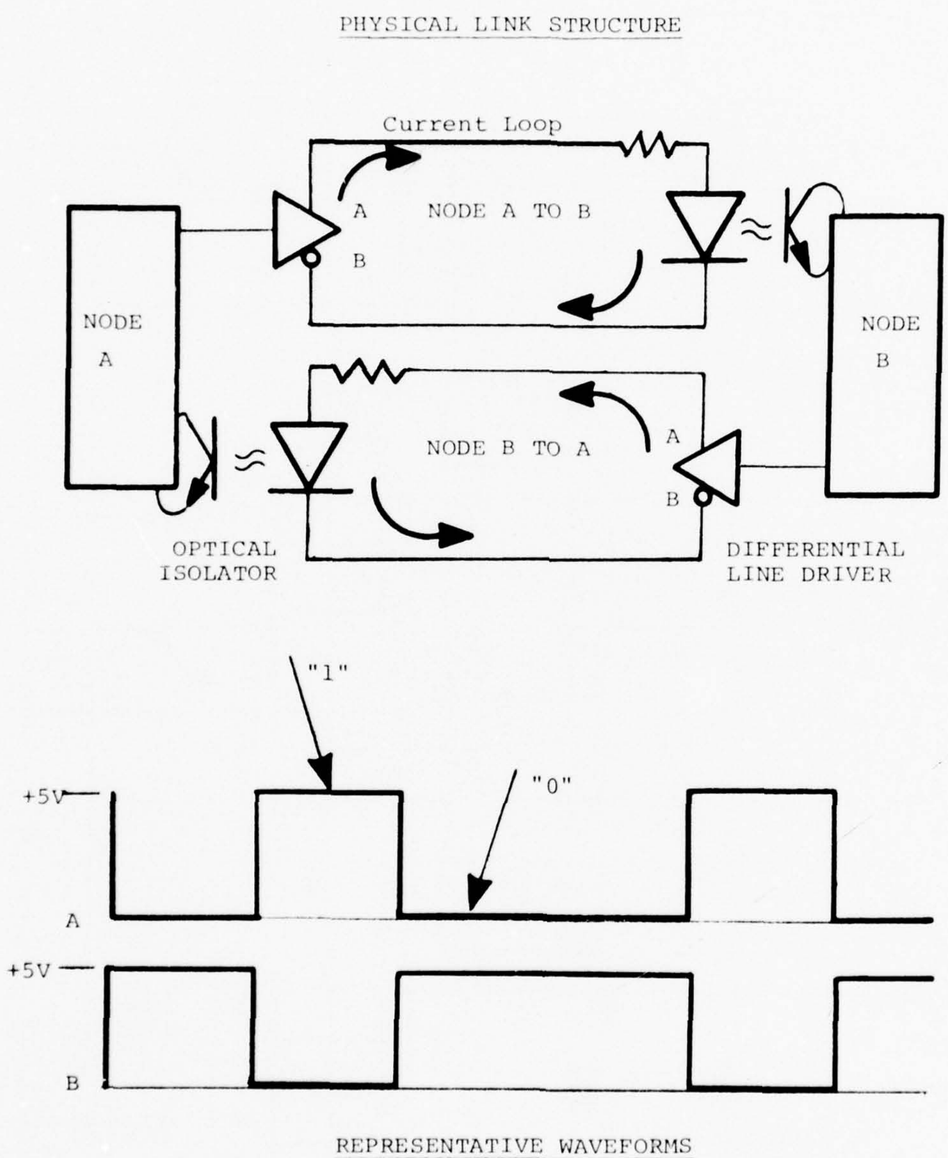


Figure 3.5 Data Link Structure.

undesired condition of data looping. As a final comment to complete the description of the data portion of th node, once a message has reached its destination, the response, if any, re-traces the same path, over which it was transmitted, back to the CPC. This is done by utilizing the other half of each duplex link involved.

3.4 Nodal Operating System

The operating system for the microprocessor node was designed and developed, in a large part, by C.J. Smith [28]. Its description is essential to the understanding of the single level network, and so a summary of some of the work of Reference 28 will be presented.

The nodal operating system is responsible for many of the same network functions as is a typical Programmable Front End Processor (PFEP) in a telecommunications network [22]. Specifically, the operating system of the M6800 microprocessor node provides for:

1. Configuration of I/O ports.
2. Error Control.
3. Message assembly.
4. Message buffering to a limited extent.
5. Code conversion and reformatting of network messages.
6. Data manipulation.

To implement the above list of capabilities, the operating system, in its current form, performs the following operations [28]:

1. Supports the circuit-switching functions.
2. Allows dynamic network reconfiguration.
3. Supports multiple I/O ports and devices.
4. Provides for error detection and error recovery procedures.
5. Utilizes full-duplex transmission links to the maximum extent possible.
6. Provides a convenient interface for background application programs.
7. Can be used in any node with any application program without requiring change.

The major portion of the operating system is divided into four sections, corresponding to the four types of interrupts recognized by the M6800 MPU: RESET, non-maskable interrupt (NMI), software interrupt (SWI), and interrupt request (IRQ). The SWI and NMI sections are completely independent, while the RESET and IRQ sections are related during the node initialization process.

Since the major portion of the operating system is the interrupt request supervisor, with its associated message processing states, it will be described first. A processing state is used to designate the process which will be activated upon the occurrence of an IRQ interrupt, such as those generated by the ACIA of a particular I/O port. Each of the three ports on a node can be simultaneously sending or receving data, and so there are six possible transitions for each IRQ interrupt. When an IRQ interrupt does occur, the interrupt request supervisor polls the three ports to determine the cause of the interrupt, and therefore which servicing routine to activate. It also decides if a state change is required on the next interrupt, and if so, updates its state table accordingly. The initial state of the operating system is set by either a RESET interrupt or else by a CPC generated RESTART command. In this manner, the CPC exercises considerable control over which processes will be activated at any one time in the node. In the following four paragraphs, the features of the four interrupt handling routines will be discussed.

I. RESET and RESTART

The RESET section of the operating system is activated by either the hardware detection of the RESET line going low, or by the interpretation of a CPC generated message as being a RESTART command. In either case, the following functions are then executed:

1. All 256 bytes of RAM are cleared.
2. Address pointers for buffer management and I/O port addressing are initialized.
3. All circuit switches are reset to the NULL state by reprogramming the PIA to output all zeros for the the 6 control bits.
4. The initial values for the six state pointers are set.

5. The unique node identifier (ID) is read from the physical slot into which the node has been placed. This ID is then stored in RAM.

6. The I/O port ACIA's are initialized to the following state: 11 bit blocks (8 data bits, 1 start, 1 stop, 1 odd parity bit), divide by 16 I/O rate, transmit data register empty (TDRE) interrupt disabled, receive data register (RDRF) interrupt enabled.

7. If the display keyboard is attached, a branch is made to the DSKY program.

8. If the teletype interface is connected, the TTY ACIA is programmed for 2 stop bits, no parity bit, and the divide by 64 mode.

9. The interrupt mask is cleared and then control is passed to the background application via a "JSR" instruction.

II. NON-MASKABLE INTERRUPT (NMI)

This section is activated only when the NMI line goes low. The only hardware unit equipped with an NMI capability is the DSKY. However, before control is passed to the DSKY NMI entry point, a check is made to ensure that the DSKY is connected.

III. SOFTWARE INTERRUPT (SWI)

The SWI interrupt is used for debugging purposes only, and is activated when an SWI instruction is executed by the background job. If the TTY is attached, its monitor program will be executed, or if the DSKY is connected, its SWI routine will be branched to. If neither unit is present, no action will be taken.

IV. INTERRUPT REQUEST (IRQ)

The processing of the IRQ interrupt has been discussed to a great extent in a previous section, but the following additional points must be appended. If an I/O port did not cause the interrupt, the TTY status register is checked to see if the TTY interface is connected. If so, the contents of the registers are printed, and the TTY monitor program maintains control. If the DSKY caused the IRQ interrupt, control is passed to its IRQ entry point, which returns via

an "RTI" instruction. Finally, if the cause of the interrupt cannot be determined, an error code is set, and control returns to the background job.

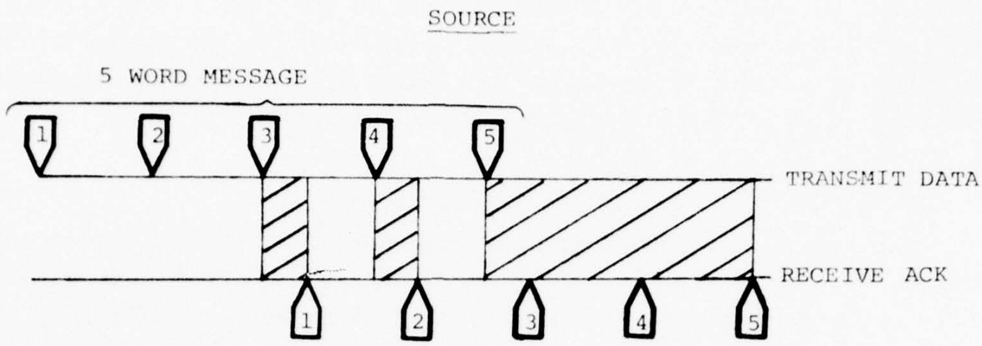
Another important portion of the operating system is the message synchronization and error control procedures. As stated before, these procedures are invoked only during DATA messages, and are not used during network control commands. In reference to Figure 3.6, the following statements concern the form of communications protocol used to implement the message synchronization.

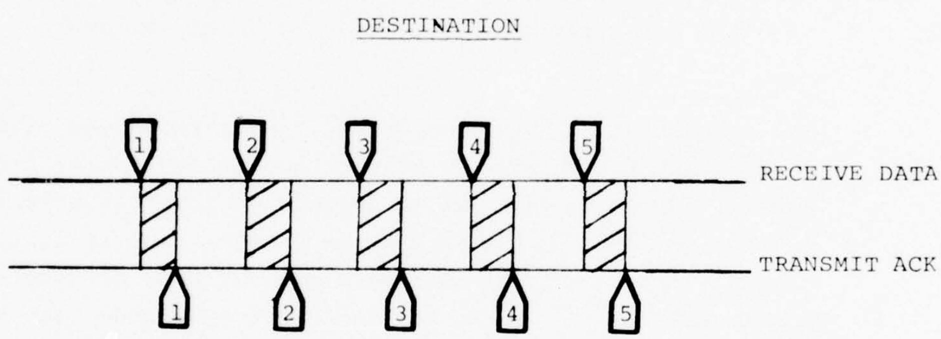
1. Each 11 bit block is edited by the receiving node's (or CPC's) ACIA. If no transmission errors are detected, an acknowledgement (ACK) is transmitted back to the message originator requesting continuation of the message.

2. Since this protocol is implemented using full-duplex links, time is not wasted by the sender while waiting for a returning ACK before sending the next block. On the contrary, up to three words can be transmitted, before the first ACK must have been received. If none is received, the transmission is terminated before a fourth word is sent. In the node, the subsequent receipt of an ACK will re-initiate transmission, while in the CPC a software timeout will occur at which time error recovery procedures will be invoked.

3. CONTROL and GATEMAN message formats (as were previously mentioned) do not use the message synchronization procedure. This is because the network control commands are themselves used in the error recovery process, and consequently, it is not possible to use them to recover from their own errors.

With the message synchronization process in mind, the error recovery routine for transmission errors will now be discussed as the final important aspect of the operating system. Upon detection of an error, the receiving node will transmit to the sender (CPC) a word indicating the nature of the error. The sender, in response, formats and sends a CONTROL message header, followed by a re-transmission of the incorrectly received data, and then the remainder of the message. If error recovery is not successful after two attempts in re-establishing communications, the network DETECT/RECONFIGURE test (which





KEY POINTS

- Source Node (or CPC) transmits up to 3 words at start.
- ACK for Word 1 must be received before Word 4 is sent; if received, message is continued.
- CONTROL messages are not acknowledged by this method.

Figure 3.6 CPC-NODE Message Synchronization.

will described in Section 3.6) will be invoked. Finally, the node will not attempt to re-establish communications on its own, but will wait for instructions from the CPC. There is no limit to the number of successful error recoveries during a single message, however, excessive error recoveries do degrade system throughput and performance.

3.5 Network Configuration and Control Software

The set of algorithms grouped together under the heading "configuration and control software" form the major contributions of the author to the development of the single level network. These programs, written in IMP-16C assembly language, reside in the main memory and in the ground support processor of the central processing center (CARDS). They consist of five basic programs, three of which can be executed as a task in a triad of processors operating in a multiprocessing configuration. A brief description of the programs is as follows:

1. GROW

   The program which sets the circuit switches of the various nodes in such a manner as to construct a funtioning I/O system. GROW must be called during the initialization process not only to construct the network, but to initialize the data base used in programs 2 and 3. GROW constructs a network, basically, by starting at a root node (the CPC), and then attempting to activate all possible links subject to the restriction that a node may not have more than one port which is INBOARD.

2. RECONFIGURE

   A derivative of GROW, called when the network must be restructured to avoid a known link or node failure. RECONFIGURE uses the same "growth" type process as 1, but relies on a data base that has been structured to only test and enable those links necessary in order to affect a network repair. It will not disturb those portions of the network which are performing normally.

3. TEST

   A relatively short program used to isolate and identify a link or node failure. TEST sends a STATUS request to each node in the net, in the order in which that node was joined

to the network. In doing this, a failure to receive a status response back from a test node can isolate a fault to a single point. TEST also verifies that the status of each port of the test node matches that found in the data base's PORT STATUS TABLE.

4. MONITOR

A program designed to be executed in the ground support processor of the CPC which continually loops through the TEST program waiting for a network failure. When a failure is detected, MONITOR branches to RECONFIGURE for correction. MONITOR also re-GROW's the network after a user selected number of TEST loops in order to reconnect failed links that have become operational since the last time GROW was called.

5. SYSPROG

An interactive demonstration program running in the ground support processor which contains programs 1 - 4 plus application subroutines for sending and receiving teletype messages. SYSPROG's major benefit, however, is that it also contains the DECSCOPE network display program and a time calculation subroutine. In this manner, not only can data on various network configurations be taken, but the dynamic reconfiguration process can be observed (refer to Figures 3.13 - 3.14).

GROW, RECONFIGURE, and TEST all run as a task in a lock-step manner in a triad of processors where the results of every instruction are voted upon. Thus, the reliability of the network is greater than it would have been if the configuration and control software had been resident in the individual nodes' operating systems. Again, as mentioned earlier, this is one of the more significant advantages of the hierarchical network design.

Before the network algorithms can be described, three preparatory topics must be addressed: the organization of the network's data base, the sizes of the different program segments, and a brief description of the six service routines (GATEMAN, CONTROL, ENDFIND, NULLFIND, UPDATE, and STATUS).

The network's data base and associated shared constants are located exclusively in the CPC's main memory. The major tables and lists are as follows:

1. PHYSICAL CONNECTION TABLE (CONTAB)

   Fifty-five sixteen bit words organized in increasing order describe the actual physical topology of the network. They are of the form (NNNN PPPP NNNN PPPP) where the first byte is the origin's node and port ID which is connected to the second byte's port and node ID or the destination of that particular link. To illustrate, the entries for node 1 are as follows:

$addr_{16}$	$contents_{16}$	
133C	1101	; port 1 to node 0, port 1
133D	1223	; port 2 to node 2, port 3
133E	1333	; port 3 to node 3, port 3
133F	0000	; future expansion to
1340	0000	; five ports/node.

2. PORT STATUS TABLE (PORTAB)

   Again as in 1, fifty-five sixteen bit words are organized by node and port in increasing order as in the CONTAB, but this time each word indicates the current state of each particular port. The PORTAB is the most important table of the network's data base for it acts as a "virtual" network in itself. The table is referenced by all configuration and control programs. PORTAB is initialized by the GROW program and then updated as the network proceeds through the growth process. The allowable status table entries are:

0000	-	null port
0001	-	inboard port
0002	-	outboard port
8000	-	failed port

3. <u>GROWLIST</u>

Sixteen consecutive sixteen bit words which indicate the order in which successive nodes accepted an INBOARD port, and became a member of the network. The GROWLIST is formed by the GROW routine, and it is an integral part of the test and reconfiguration data base.

4. <u>RESET LIST</u>

Six consecutive sixteen bit words which indicate the order in which to prepare a damaged network for reconfiguration. More explanation of the RESET LIST will given in a later section.

Providing essential functions for the GROW and RECONFIGURE routines are the following service subroutines:

1. <u>GATEMAN</u>

Routine which makes the node and port whose ID's are passed to it in registers one and three, an INBOARD port. GATEMAN returns to the main program a 0 if successful and a 1 otherwise.

2. <u>CONTROL</u>

Routine which makes the node and port whose ID's are passed to it in registers one and three, an OUTBOARD port if register zero = 0, or a NULL port if register zero = 1. CONTROL returns to the main program a 0 if successful and a 1 otherwise.

3. <u>STATUS</u>

Routine which verifies that GATEMAN or CONTROL were successful in setting the state of a particular port. STATUS returns to the main program a 0 if successful and a 1 otherwise.

4. UPDATE

Routine which uses the node and port ID's passed to it in registers one and three, and the status word passed in register zero to update a particular entry in the PORT STATUS TABLE.

5. ENDFIND

Routine which searches the CONNECTION TABLE for the node and port, whose ID's are passed in registers one and three, for the termination of the link, whose starting point is indicated by the given ID's. If successful, ENDFIND returns to the main program the ENDPOINT ID's in registers one and three, otherwise ENDFIND restores registers one and three with the original ID's.

6. NULLFIND

Routine which searches the PORT STATUS TABLE for an available NULL port from which a possible new link can be originated. NULLFIND is subject to two constraints. If called during GROW, the NULL port ID returned must be from a possible link whose temination node does not already have an INBOARD port. If called during RECONFIGURE, the NULL port ID returned must be on a node which does already have an INBOARD port . These two seemingly conflicting statements will be made more clear in the next subsection.

To conclude the preparatory remarks, Table 3 displays the relative sizes of the various configuration and control program segments.

TABLE 3

CONFIGURATION AND CONTROL

SEGMENT LENGTHS

SEGMENT	LOCATION	# OF WORDS
Data base and shared constants	Main memory	317
Service subroutines	Main memory	135
TEST routine	Main memory	139
GROW/RECONFIGURE routines	Main memory	495
SYSPROG (inter-active DECSCOPE program)	Ground support processor	714

3.5.1 GROW

The GROW routine establishes the initial state of the I/O network by attempting to activate a path to every node. The principle input to the GROW routine is the CONNECTION TABLE and the principle output , besides the physical setting of the network circuit switches, is the PORT STATUS TABLE. Prior error and status information is not required for the GROW routine, for it will circumvent any failed links or nodes as it unsuccessfully attempts to transmit GATEMAN or CONTROL commands to the faulty elements.

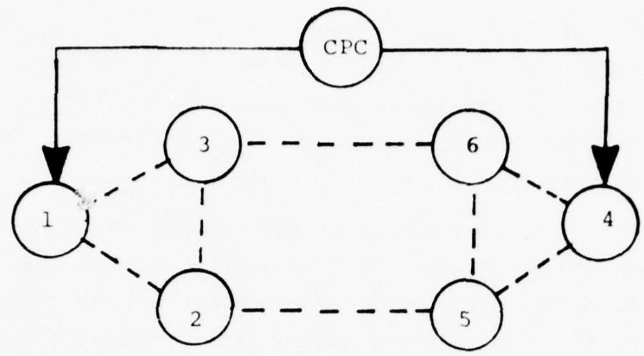
Upon initiation, GROW records the system time for data taking purposes,and initializes all tables and variables. It then sets the central processing element as the root and initial "GROW-NODE" for the growth process. The node and port ID from which a link is to be grown is called the "GROW-POINT". The first GROW-POINT to be considered on any node is always port 1. Therefore, the CPC's port 1 is designated as the initial GROW-POINT. Next, ENDFIND is called to determine if the GROW-POINT has a valid termination or "END-POINT". Since it does,a GATEMAN command is sent to the END-POINT to make it an INBOARD port. If GATEMAN is successful, the ID of the END-NODE is placed on the "first-in-first-out" (FIFO) GROWLIST. This node will be the next GROW-NODE to be considered. The value of the node counter "NODENUM" is also incremented at this point. At the completion of the GROW routine, this counter will be compared to the actual number of physical nodes in the network. In this way a rapid determination can be made as to the overall connectivity of the network.

Since a link has now been constructed, the current GROW-POINT is exhausted. Consequently, NULLFIND is called to determine the next GROW-POINT, if any, on the current GROW-NODE. In the case of the CPC, two GROW-POINTS are possible and so the CPC's port 2 becomes the next GROW-POINT. The linking process is now repeated and, if successful, a second node is placed on the GROWLIST and NODENUM is again incremented. A snapshot of the network so far is:

GROWLIST

1

4



Proceeding ahead, the current GROW-NODE is again checked by NULLFIND for the next available NULL port. Since none exist, the next GROW-NODE is fetched from the GROWLIST and the GROWLIST pointer is incremented. The linking process is now repeated for node 1, port 1, identically to that used for node 0 (the CPC) with one exception. Before a link can be grown from a GROW-POINT the actual circuit switch must be made OUTBOARD using a network CONTROL command. Also, as explained for the NULLFIND subroutine, a link will not be grown to a node which already has an INBOARD port. By applying the procedures outlined so far to node 1, the following is the network status at the point where node 1 has exhausted all of its GROW-POINTS:

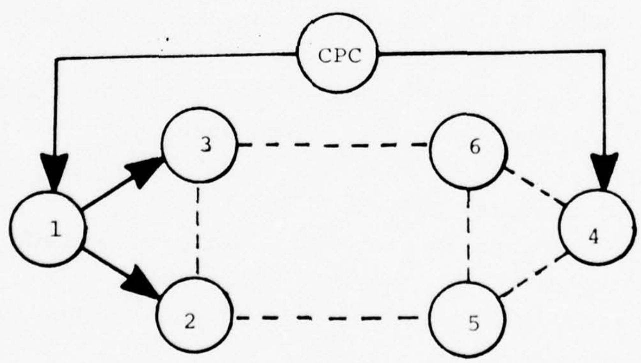
GROWLIST

1

4

2

3



From this point, the operation of the GROW program proceeds rapidly to connect nodes 5 and 6. Again, all nodes who accept an INBOARD port are placed on the GROWLIST, and must be utilized as a valid GROW-NODE. When a value of 0 is read as the GROW-NODE, the GROWLIST has been exhausted and the GROW routine can proceed no

further.

To complete the process a check is made to see if NODENUM=6. If so, all six nodes have been made a part of the network. The GROW program also records the terminal time and computes the elapsed time for the growth process before retiring. (For a detailed description of the GROW algorithm see Figure 3.7).

Although an extensive study of GROW-TIMES under various initial conditions is presented in Chapter 6, three examples are displayed in Figures 3.8, 3.9, 3.10. Notice the variations in the three configurations' GROW-TIMES. The variations are due primarily to differences in total I/O time required in these three examples. Again, this will be amplified in Chapter 6. As a final comment, GROW is also called periodically, by other routines such as TEST or SYSPROG, to re-GROW the network. In this way contact can be re-established to nodes which may have become operational again after previously being isolated from the network during the fault detection process.

3.5.2 RECONFIGURE

The RECONFIGURE routine is a derivative of the GROW routine. It is passed by TEST, the node and port ID of the END-POINT from which no STATUS request response was received. RECONFIGURE begins by marking both ends of this link FAILED in the PORT STATUS TABLE. The PORT STATUS TABLE and GROWLIST up to this point are in the same state as when the GROW routine had finished some time previously. RECONFIGURE must now RESET the PORT STATUS TABLE to reflect that a portion of the network previously connected has been isolated, when the faulty link ceased to function. Once the PORT STATUS TABLE has been RESET, RECONFIGURE simply sets its GROW-NODE pointer to the first node on the GROWLIST and then branches to the GROW Routine to proceed as in a partially completed network. The RESET process, however, is not a straightforward procedure as can be quickly seen by the flowchart of the RECONFIGURE routine (Figure 3.11).

The difficulty of RESETTING the PORT STATUS TABLE can be demonstrated by considering the following example.

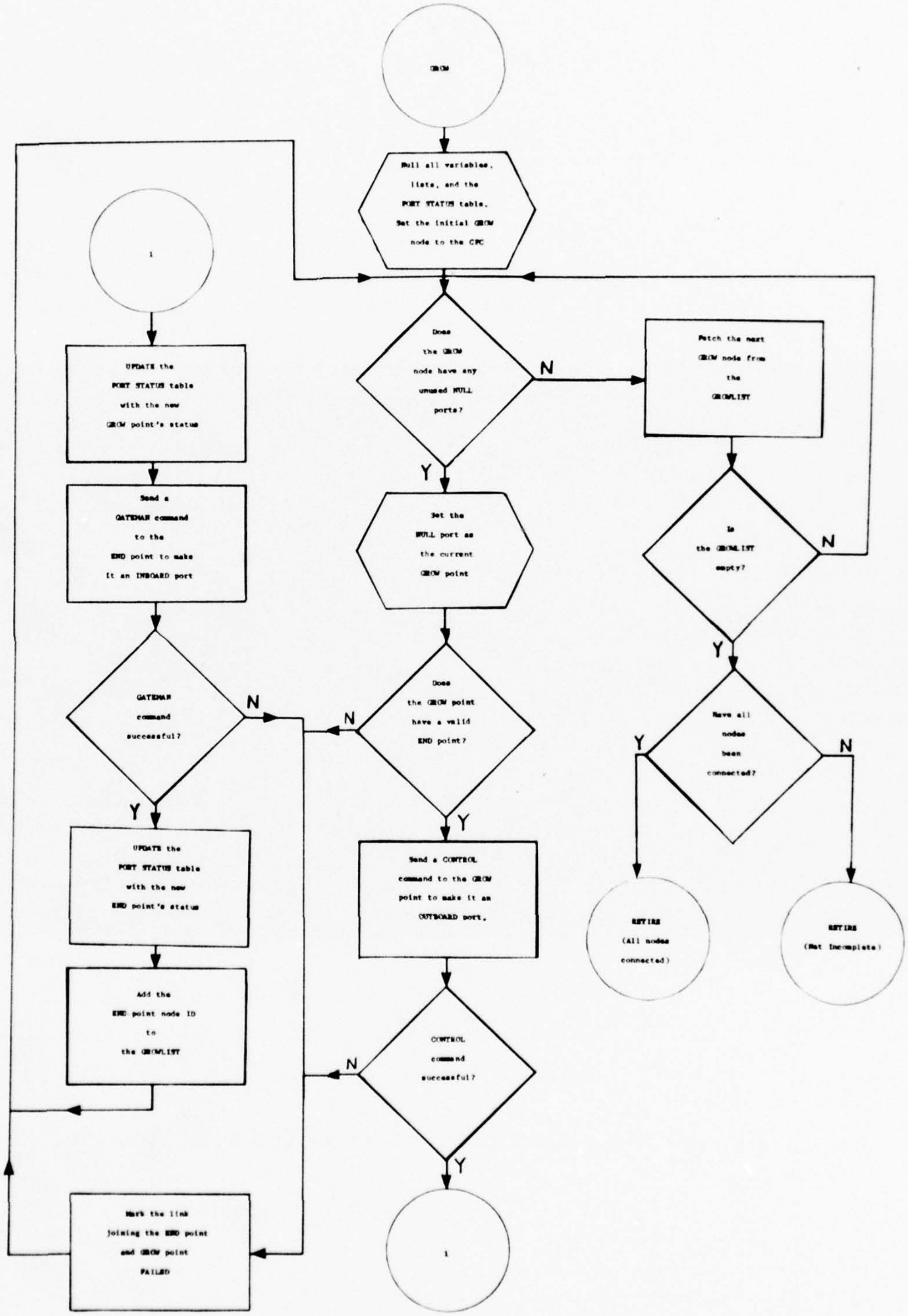


Figure 3.7 Flowchart of the GROW Routine.

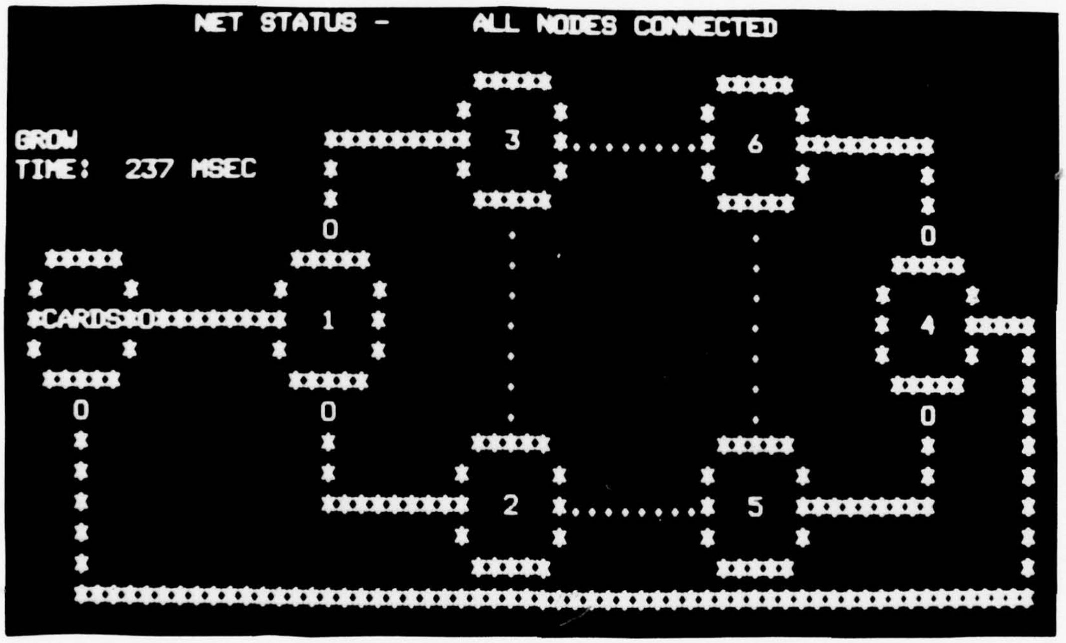


Figure 3.8 Results of GROW for a Typical Six-Node Network.

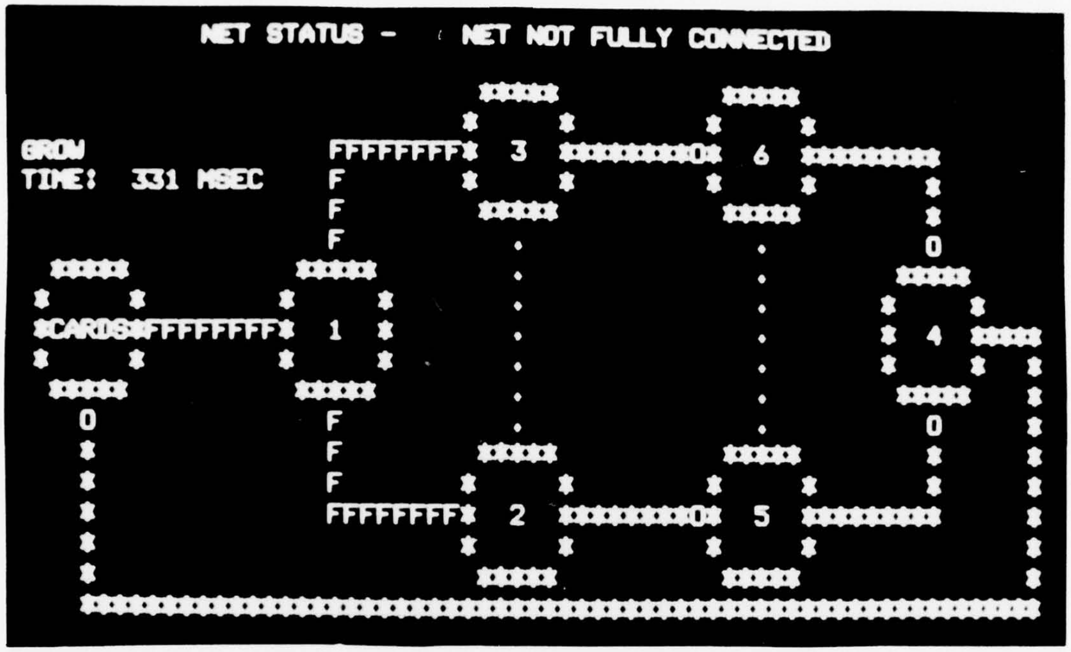


Figure 3.9 Results of GROW for Six-Node Network with One Node Failure.

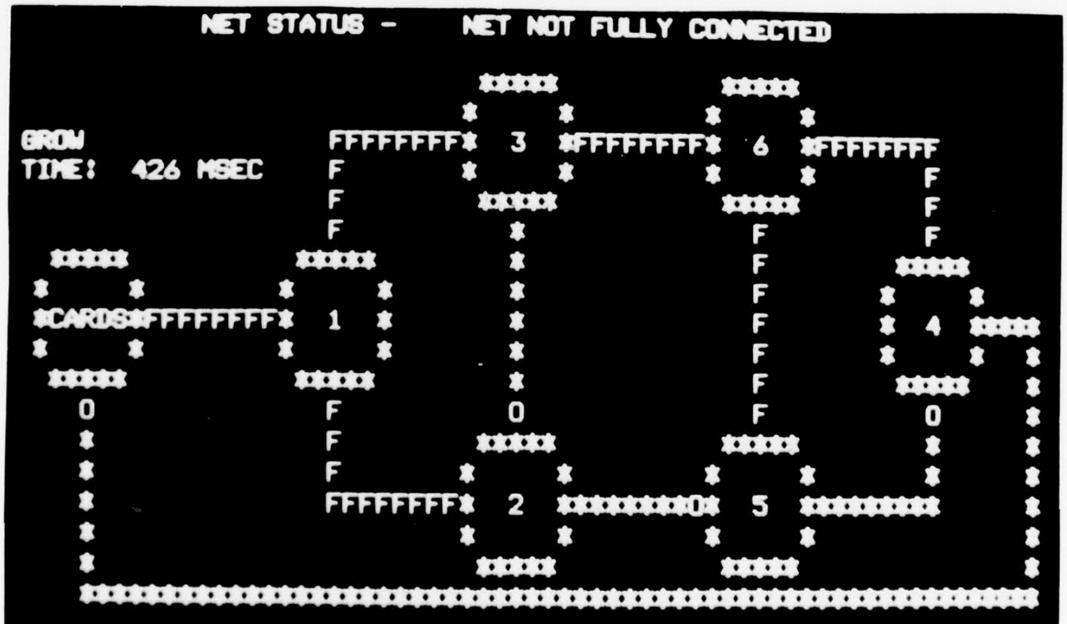
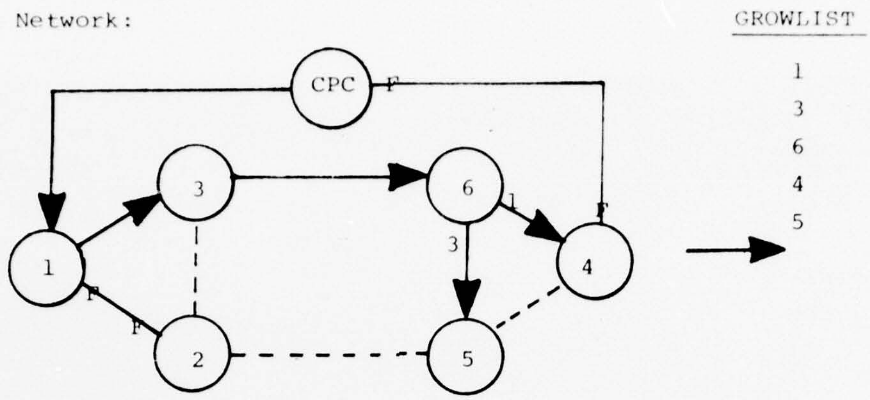


Figure 3.10 Results of GROW for Six-Node Network with Two Node Failures.

1. Given the Network:



Links CPC-4, and 1-2 have failed and links 2-3, 2-5, and 4-5 are spares.

2. RECONFIGURE the network given that:

Link 3-6 has failed.

3. Solution:

It can be seen quickly that nodes 4,5, and 6 have been left dangling by the given link failure. In response to the failure, link 3-6 is marked FAILED, and RESET then proceeds to modify the PORT STATUS TABLE and the GROWLIST in the following manner: (again refer to Figure 3.11).

a. Since the FAILED-NODE, node 6, has 2 OUTBOARD ports, a trace must be made of each "branch" to find its terminal point. Once a termination has been found all entries for the terminal node must be nulled in the PORT STATUS TABLE and the terminal node's ID removed from the GROWLIST. (This is the essence of the RESET process.)

b. Next, the trace is "unwound" back towards the FAILED-NODE, repeating the expulsion process for each node in the path.

c. When the FAILED-NODE is again reached, a trace is made of the other OUTBOARD port. Similar initializing procedures are carried out for the nodes in its path also.

d. Note, difficulties arise when a node in the trace has more than one OUTBOARD port resulting in multiple terminations. RESET handles this condition easily through the use of a software stack called the RESET LIST. The RESET LIST

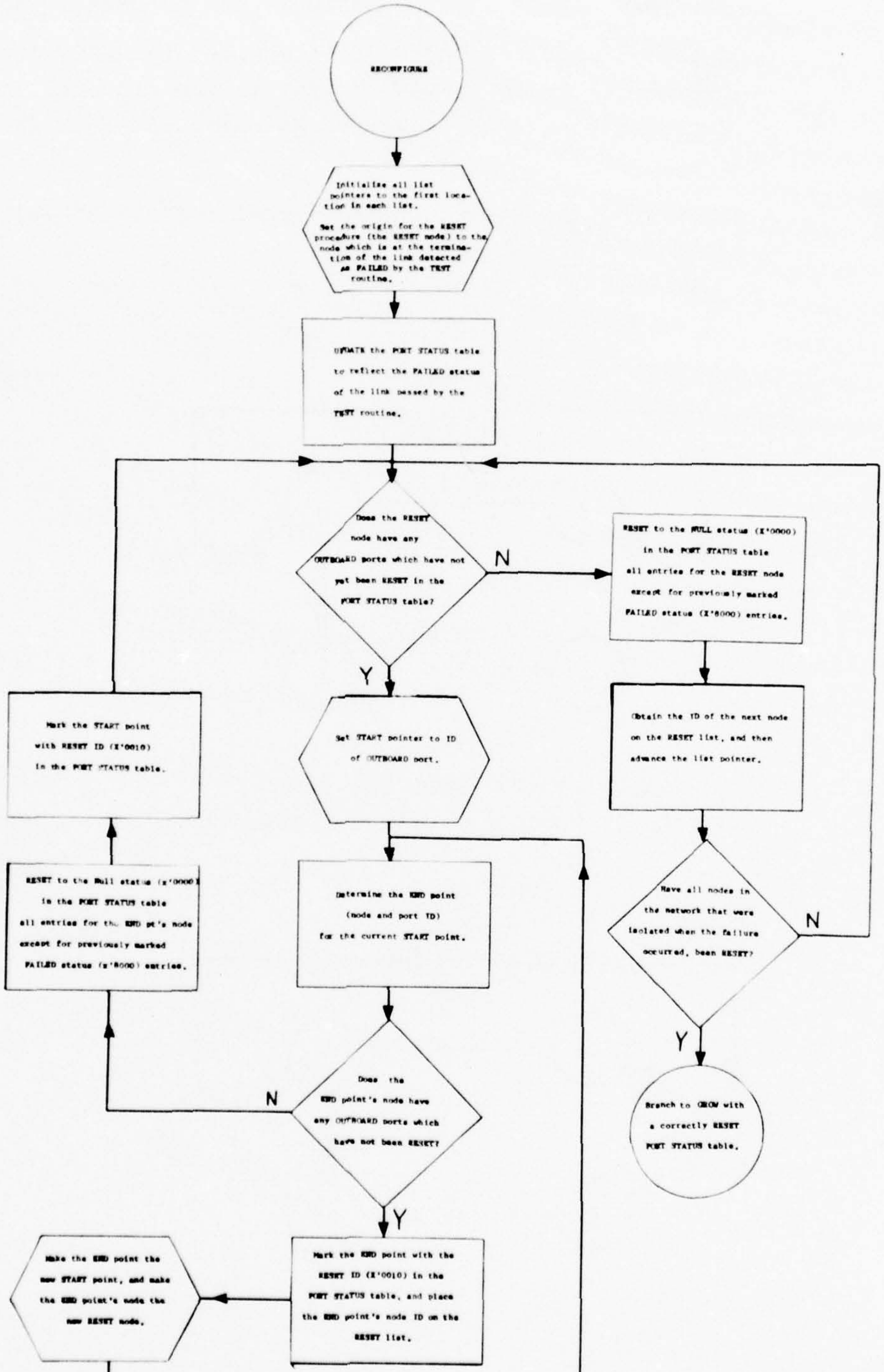


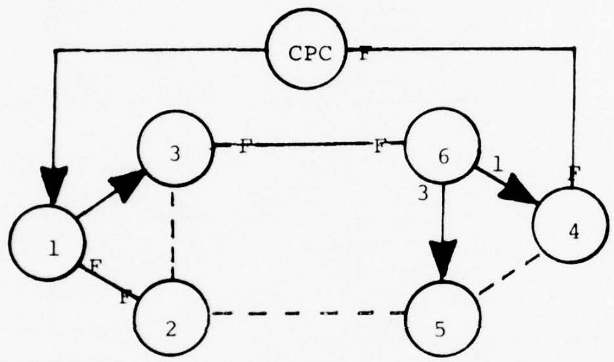
Figure 3.11 Flowchart of the RECONFIGURE Routine.

implements a FIFO scheme similar to the GROWLIST to effectively handle all the nodes in the network to be NULLed.

e. Once the original FAILED-NODE is completely RESET (both traces complete and the RESET LIST is empty), the RECONFIGURE program branches to an entry point in the GROW routine.

f. A summary of the steps outlined above for the example presented is as follows:

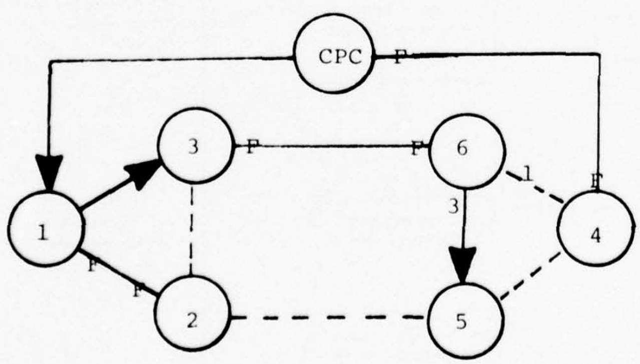
(1) LINK 3-6 HAS FAILED



GROWLIST

1
3
6
4
5

(2) NETWORK STATUS AFTER THE RESET OF NODE 6, PORT 1 (Results of Trace 1)

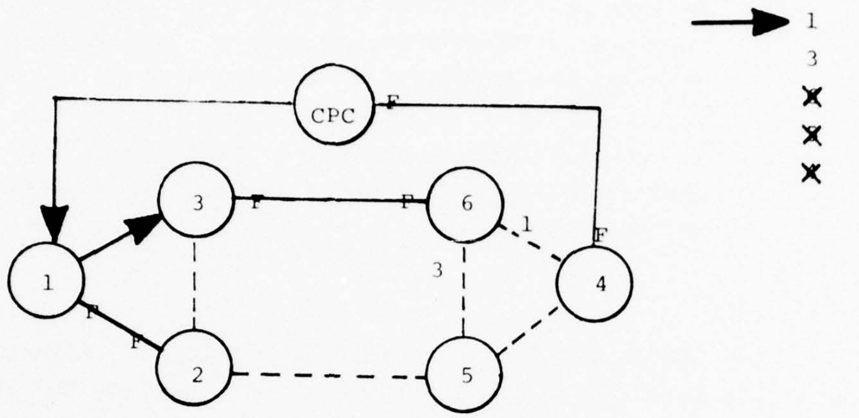


GROWLIST

1
3
6
~~4~~
5

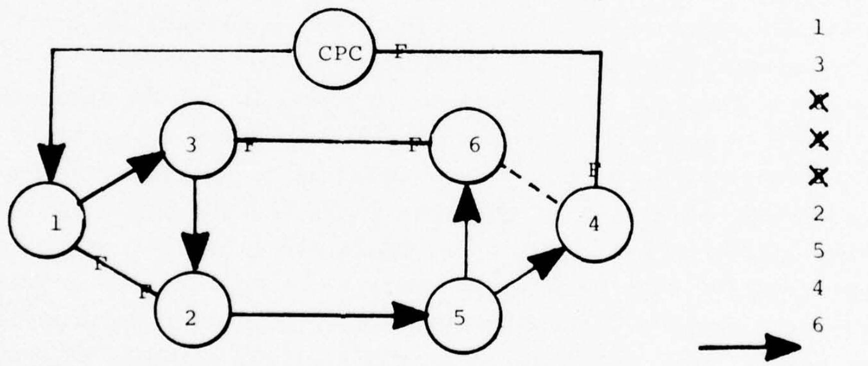
(3) NETWORK STATUS AFTER THE RESET OF NODE 6, PORT 3 (results of Trace 2), (also status at entry to GROW routine)

GROWLIST



(4) STATUS OF FINAL RECONFIGURED NETWORK after completion of GROW ROUTINE.

GROWLIST



g. Notice that in the above example, that only those links that were essential to the RECONFIGURE process were activated. Also, note that the GROWLIST has been restructured so that it again reflects the order in which the nodes joined the network.

h. As a means of comparison, RECONFIGURE was initially implemented similar to an algorithm found in [25] that was used in the earlier Draper Fault-Tolerant Network Effort. Initial results have demonstrated that the utilization of the RESET concept have resulted in an approximately 40%

savings in RECONFIGURATION-TIME. This fact is further demonstrated by the series of DECSCOPE photographs, Figures 3.13 through 3.14. In this example, the correction of a link failure by RECONFIGURE was four times faster than that obtained by RE-GROWING the network. This startling result demonstrates the value of the RECONFIGURE routine, and it emphasizes that minimizing the number of links that have to be de-activated re-activated, lessens the time lost by an application process such as the digital autopilot, during the RECONFIGURATION task.

3.5.3 TEST

The TEST routine is the fault detection and isolation program for the single level network. (A flowchart of its algorithm is given in Figure 3.12). TEST's relatively short length is indicative of the fact that it is a straightforward program that attempts to verify that communications exist between the CPC and every node of the network. It facilitates this by transmitting a STATUS request to each node whose ID is on the GROWLIST, in the order in which that node was added to the network. This is a crucial point, for it allows the non-receipt of a single STATUS request to pinpoint a FAILED link down to one possible choice. This is a valid procedure since the links were added to the network in much the same way that a tree grows. Verifying that all the n-1 links are good, which were added up to the point where the nth link is tested, says that if the test fails for the nth link, it must be that link which has malfunctioned. Therefore the order of the node ID's on the GROWLIST must correspond exactly to the sequence in which the network was GROWN or RECONFIGURED. TEST also performs a verification function. By comparing the STATUS received from the TEST-NODE with the entries in the more reliable "virtual" network of the PORT STATUS TABLE, discrepancies can be detected and corrected. Since the PORT STATUS TABLE is deemed correct, a discrepancy is treated like a link or node failure, and results in the RECONFIGURE routine being called. The TEST program runs either in the demonstration MONITOR mode or by the normal process of being invoked as a task when error recovery procedures have not been successful (see Section 3.6). TEST in this case, detects the error and passes the information to the RECONFIGURE routine for further processing.

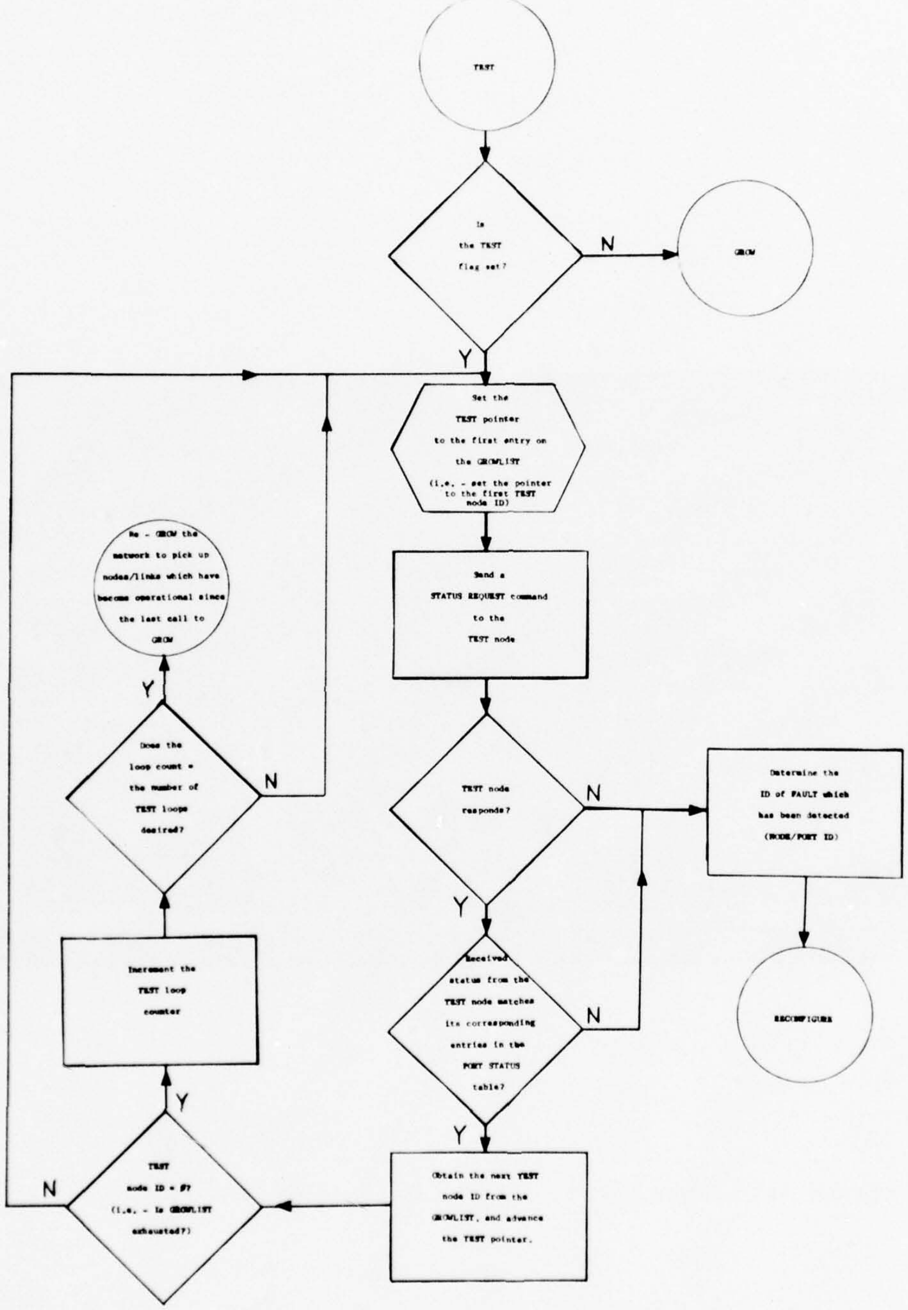


Figure 3.12 Flowchart of the TEST Routine.

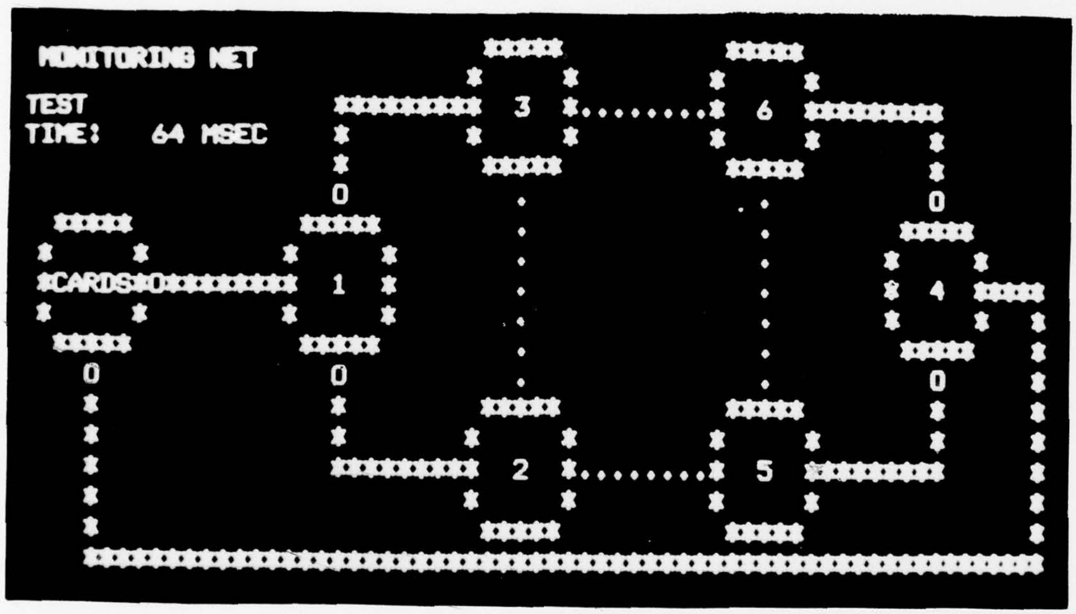
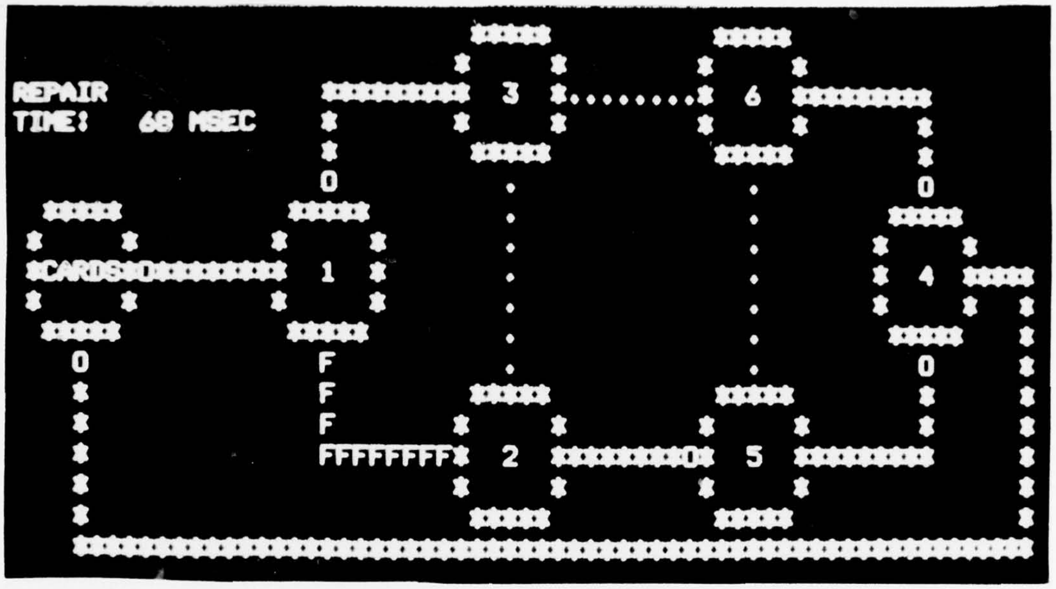
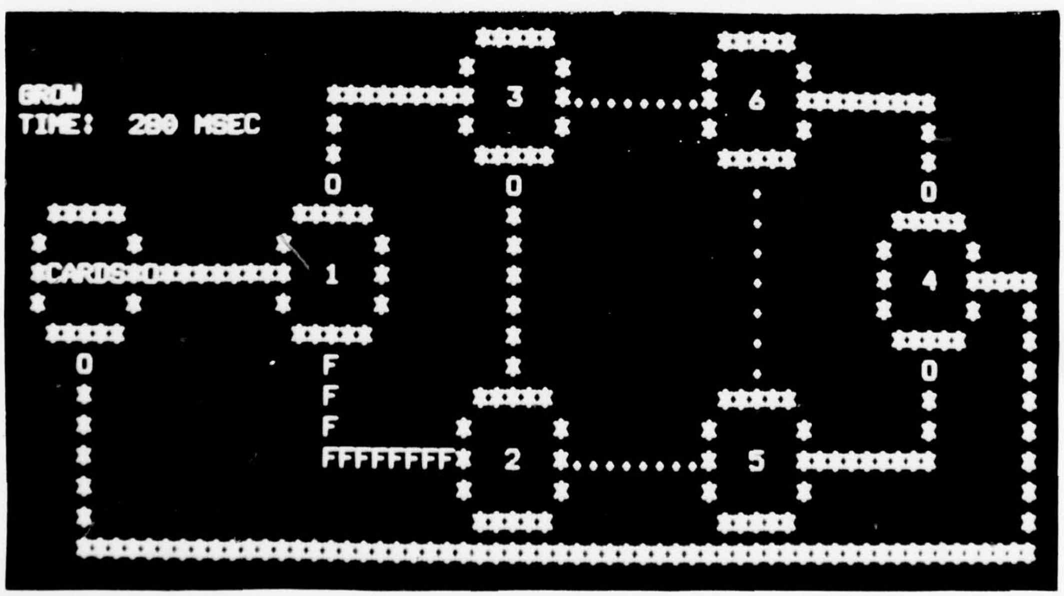


Figure 3.13 Monitoring a Six-Node Network for Faults.



Link 1-2 Failure Detected by TEST and Network
Repaired by RECONFIGURE.



Link 1-2 Failure Repaired by RE-GROW.

Figure 3.14 Comparison of RECONFIGURE to RE-GROW
for Network Repair.

3.5.4 SYSPROG

This interactive program,stored in the PROM of the ground support processor,is designed to be a demonstration and data collection tool for use with the single level and eventually with the bilevel network. It does not interfere with the operation of programs executing in the multiprocessor triads. It can, however, load from the main memory the same GROW, RECONFIGURE, and TEST programs into the cache memory of the ground support processor for debugging purposes. The following DECSCOPE commands are available to the operator desiring to utilize SYSPROG:

1. INIT

   This command clears the DECSCOPE network display, initializes all variables, tables, and constants used by SYSPROG. INIT places SYSPROG into the command interpretation state awaiting the next instruction.

2. GROW

   This command causes a branch to the GROW routine which has been loaded into the ground support processor's cache memory. GROW also displays the results of the GROW routine on the DECSCOPE.

3. MONITOR

   This command places the network control into a test loop awaiting a link or node failure to be detected by the TEST routine. When TEST detects a fault,control automatically is passed to the RECONFIGURE routine for correction. Upon completion, the results of reconfiguration are displayed and the test loop is once again entered.

4. SEND/GET

   These two complementary commands are used to send and receive teletype messages via the DECSCPOE command line. The destination node at which the teletype interface is attached must be specified prior to the actual message transmission.

5. SYSPROG "TUNING" FEATURES

   a.FASTER/SLOWER

These commands control the speed at which the GROW routine's results are displayed. FASTER displays the network status only after the GROW routine has been completed, while SLOWER displays each link as it is activated.

b. LOOPS

This command specifies the number of test loops desired during a MONITOR operation before the network is RE-GROWN. LOOPS allows the network to become more or less sensitive to transient faults, by being able to vary the rate at which a previously failed link may re-join the network subsequent to the link becoming operational again.

c. DELAY

This command allows a variable number of loops in the frequently called DELAY subroutine. The most significant use of this feature is in GATEMAN subroutine where an important amount of delay is placed between the transmission of a RESET and GATEMAN command. This amount of delay must be variable to provide for efficient network operation, since the time required to process network commands varies between the single level and bilevel nodes.

6. GOTO,FDIR,STATUS

These three commands allow branching to any address in the ground support processor. In the case of FDIR and STATUS, control is passed to the entry points of the other two system status display programs, while GOTO allows an arbitrary branch to the address which is appended to it when it is typed on the command line.

3.6 Incorporation of the Network Configuration and Control Software into the CARDS Multiprocessor

One of the most important benefits of the OSIRIS hierarchical network structure, as has been cited several times, is that the recon-

figuration and control algorithms are executed normally in the triply redundant,fault-tolerant environment of the CARDS multiprocessor.With this feature in mind, the configuration and control programs were integrated into the overall OSIRIS system in the following manner:

1. They are invoked by the system I/O routines when error recovery procedures are unsuccessful.

2. They are invoked on a periodic basis by a system task designed to detect latent faults in portions of the network not currently involved in I/O.

Since the GROW, RECONFIGURE, and TEST routines were originally executed in the ground support processor for debugging purposes, several modifications were required to adapt them so that they could be run as a task in a triad of processors. The following lists the more important of these changes:

1. The three programs were separated from the associated test code and assembled together in the format required of a system task. Since the total length of the network control programs was slightly less than the 768 words alloted in a processor cache memory for a task and its software stack, only one task was created. This task was given a unique ID and the name DETECT/RECONFIGURE task.

2. The DETECT/RECONFIGURE task was stored in an assigned address space in main memory which was not used by any other task. A pointer to its entry point was placed in the Task Identification Table located in the base page of main memory.

3. All references to the display portion of SYSPROG still remaining in the DETECT/RECONFIGURE task were removed.There can be no transfer of control outside the triad while it is running.

4. The task was made fully relocatable by removing all references to specific memory locations and by using indexed addressing wherever appropriate.

5. All constants used by the task were placed in the base page of the processor's cache memory. However, upon initiation, these constants required initialization which was accomplished by reading into the base page a block of code containing the

values of the constants. This procedure, while time consuming, was necessary since each task invoked into a processor uses the base page for its own variables and constants and consequently does not preserve the data there from a previous task.

6. Finally, all system variables used both by SYSPROG and the DETECT/RECONFIGURE task were placed in main memory, a shared resource. In this way interaction between the task and the control program was facilitated. Specifically, configuration speeds, and the number of test loops desired before RE-GROW, could still be controlled by the operator at the DECSCOPE, in spite of the configuration and control algorithms being executed in a multiprocessing environment.

Even when all of the above procedures are carried out properly, DETECT/RECONFIGURE will not be executed as a task unless it is added to the Time Event Queue of the multiprocessor. The Time Event Queue is the scheduler for the various system tasks. It controls the order of execution, execution times, iteration rates, triad assignments, and execution restrictions, if any. The queue, as diagrammed in Figure 3.15, joins the set of tasks using a series of chained address pointers.

Since the TEST routine requires approximately 70 msec. for a typical six-node network, an iteration rate of 1 Hertz was chosen for the network task. In this way, the network will be scanned for latent faults once every second. If a fault is detected, RECONFIGURE will be branched to for correction before the task retires. Further, if the required number of test loops, as specified by LOOPS, have been made on the previous number of task iterations, then the GROW program will be called for RE-GROWTH of the network. Again, once this is complete the task will retire. In all three possible variations of the DETECT/RECONFIGURE task the total time that normal system traffic will be suspended, has been shown to be of the order of one second (refer to Chapter 6). While this amounts to an excessive portion of the available I/O bandwidth for an iteration rate of 1 Hertz, the one second figureis more of an upper bound for infrequent occurrences. In a normal situation, no faults would be detected and so DETECT/RECONFIGURE would retire in the acceptable time of 100 msec.

The Time Event Queue

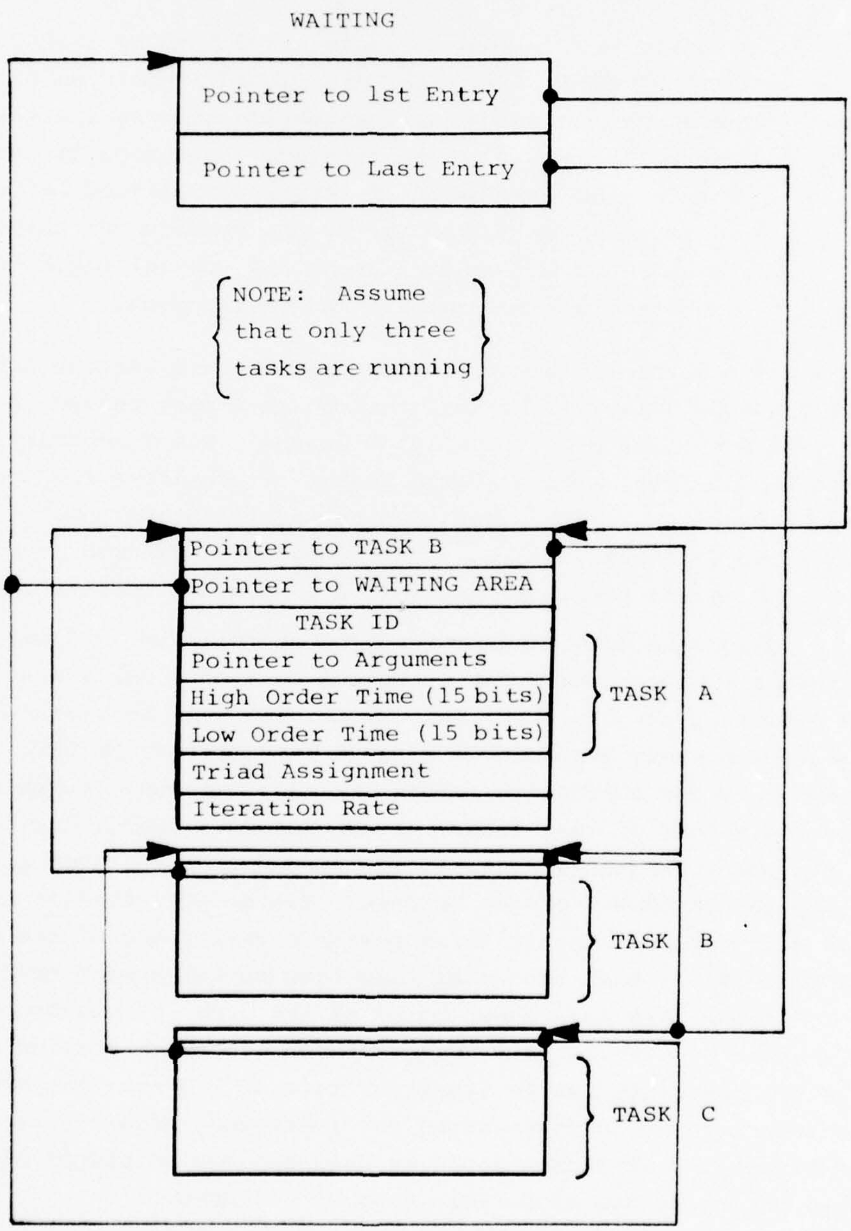


Figure 3.15 Relation of a Typical Task to the Time Event Queue.

CHAPTER 4

THE BILEVEL TEN-NODE NETWORK

4.1 General Description and Capabilities

The development of the bilevel network as an improvement to the single level network of Chapter III was undertaken primarily to further enhance the benefits that networking can provide to the area of distributed processing. The variety and often extensive number of sensors and effectors found in a typical flight control environment impose severe bandwidth and reaction time requirements on the central processing center (CPC). If this processing load can be distributed to a set of local processors, preferably co-located with the individual sensors or effectors, a great savings in bandwidth is realized [16]. Though this forms the basic justification for the single level network, as previously described, the bilevel network goes one step futher by improving the throughput of the local processors. It does this by implementing a network with two hierarchical levels, the upper level emphasizing control and data transfer, and the lower level emphasizing computation and data reduction.

The concept of a bilevel network is not new. D.W. Davies and his co-workers at the British National Physical Laboratory proposed a similar idea for use in a telecommunications environment [6]. They envisioned a two level network in which the upper level would be responsible for long distance packet transmission and switching functions, while the lower level would be a local area network serving a central phone exchange, for example. This organizing of similar processing functions into sub-networks carries over into the OSIRIS bilevel network concept. Here, the upper level network can be considered as a group of "middle manager" nodes. Each middle manager node is responsible to the CPC only, but can have subordinate to it one or more nodes forming a lower level network (see Figure 4.1). Since the middle manager node is a member of both hierarchical levels simultaneously, it is also known as a "bilevel" node. Analogous to the phone exchange

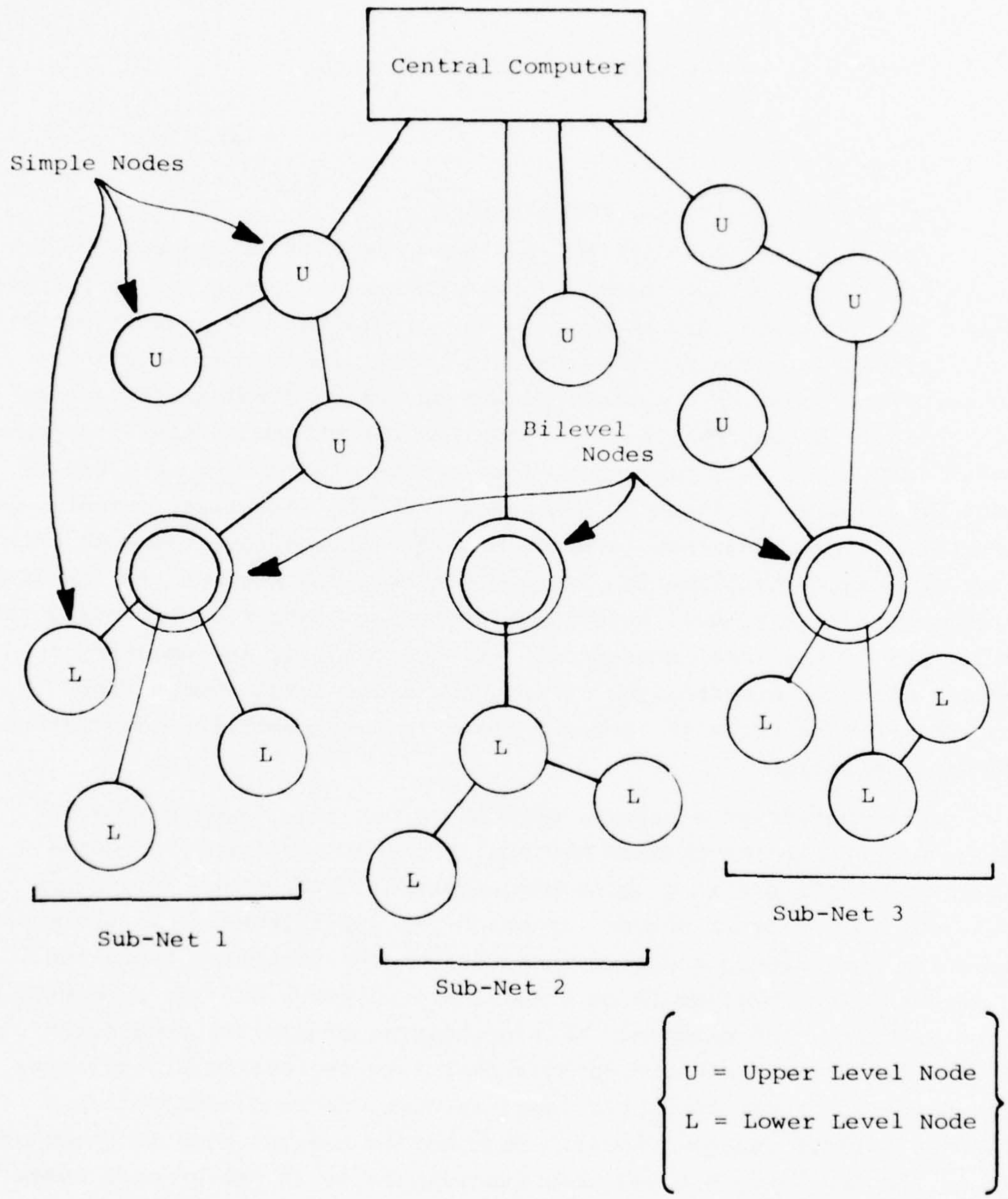


Figure 4.1 The Bilevel Network Concept.

example, each lower level network, ideally, is comprised of nodes whose attached host processors all perform a related function such as navigation.

The real benefit of the bilevel network over the simplex network, as has been stated earlier, is that it improves the computation throughput possible at a lower level processor node. It does this by not interrupting every lower level node every time a request for data is originated by the CPC. In the normal single level network each message sent by the CPC generates an IRQ interrupt in every node, which must be processed to determine if action is required. Thus, any background processing being done at a node is continually being suspended while the operating system determines the nature of the received message. In the bilevel network; however, only the upper level of the network is interrogated directly by the CPC. Each bilevel node has the ability to intercept and re-transmit all messages designated for any of the nodes of its sub-net. Only when such a message is received, is the lower level interrupted for data. In this way, the lower level network becomes a more efficient computer and spends less of its time processing data requests not intended for its use. This then is a basic justification for the bilevel network concept. As an alternative, another solution to the constant interrupt problem, investigated in Ref. 31, has been to implement a node with two microprocessors, one dedicated to background processing and one designed to handle to the communications and control functions exclusively. In this way, each dual processor node can simultaneously handle both primary nodal functions.

The experimental bilevel network at the C.S. Draper Laboratory, has the following characteristics and capabilities in addition to those listed for the single level network in Section 3.1.

1. Network Size and Composition - ten nodes, two of which can be bilevel. The increased network size adds to the richness of various topologies possible, while the implementation of just two bilevel nodes can clearly demonstrate the bilevel concept.

2. Memory Capacity - an additional 1k of RAM and 2.5k of PROM for each bilevel node.

3. Input/Output Ports - 3 additional for each bilevel node for a total of six.

4. Internal Switching Functions - an additional PIA has been added to handle the enabling bits for the extra three I/O

ports.

5. Communication method - circuit-switching, as before - internal to each network level; packet-switching of 1 byte packets at each bilevel node.

6. Number of sub-networks possible from any bilevel node - one, due to the restrictions placed on the ability of the software operating system to handle the processing of bilevel interface ports. Currently, only one interface port can be serviced for a given bilevel node.

7. Software additions - expansion of the nodal operating system to handle the bilevel control command, and additional buffer handling routines to implement the packet-switching communications scheme.

4.2 Test Network Topology Selection

The topology chosen for the bilevel network was basically the six node topology selected earlier with four additional nodes. (see Figure 4.2). Since no actual partitioning according to node processing functions was made, it was decided to use a topology which would best demonstrate the bilevel network concept. That is, each bilevel node was provided with up to four nearest neighbors which could function as members of a lower level network. In addition, two links were added joining the two sub-nets. These were implemented so that nodes could be exchanged between the sub-nets in the event of link or node failures. As in the single-level network, three of the nodes were placed in the Hybrid Simulation Facility near the aircraft flight simulator. The other seven nodes were located together in Advanced Digital Systems Laboratory near the Central Processing Center (refer to Figure 3.1 again). Finally, a DECSCOPE display routine was also written showing the ten node network in a symmetric arrangement, much like Figure 4.2, in order to facilitate the display of network status. Now that the basic characteristics of the bilevel topology have been outlined, the modifications to the existing single level software and hardware, required for the bilevel network, will be discussed.

4.3 Nodal Hardware Description

The additional hardware required to implement a bilevel node is located on a bilevel interface board. This board together with the previously described M6800 microprocessor node board, constitute

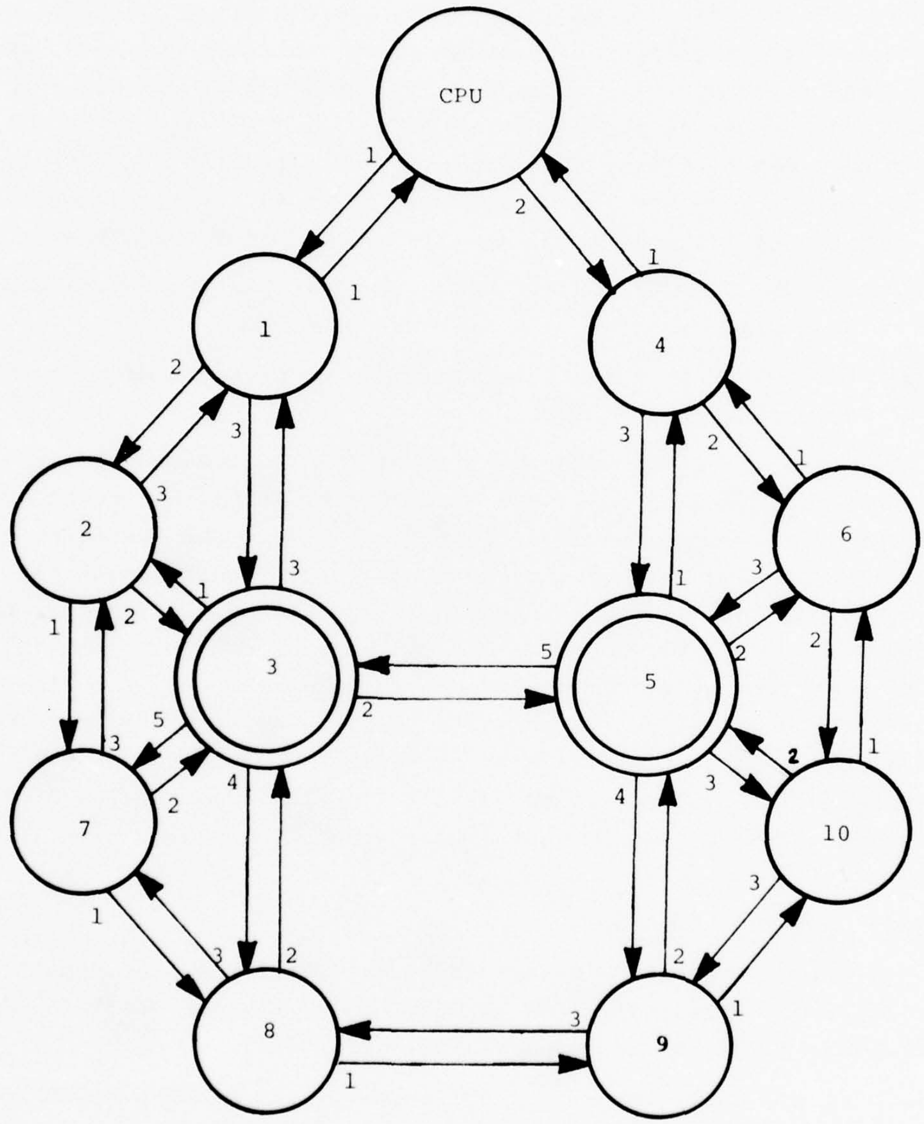


Figure 4.2 Bilevel Ten-Node Network Topology.

the complete bilevel node. The bilevel interface board is much simpler in design than the M6800 microprocessor node board. It is comprised, basically, of the additional memory required by the bilevel operating system, and the additional hardware required for the three extra I/O ports. Specifically, to add the three ports, one peripheral interface adapter (PIA) was necessary to supply the required six enabling signals. In addition, three asynchronous communication interface adapters, and a similar number of optical isolators, and differential line drivers were provided. Finally, the requisite number of gates and tri-state buffers were included to affect the same internal switching circuitry as in Figure 3.4. To add the increased memory, the following number of memory elements were appended to the node's data and address buses

1. 5 - MM5204Q 512 x 8 bit National Semiconductor Programmable Read Only Memories
2. 8 - MM2102 1k x 1 bit National Semiconductor Random Access Memories.

The actual separation in the bilevel node of the two hierarchical network levels was accomplished via software, through a new CONTROL command specifically designed for the bilevel node. The specifics of this new feature along with other additions to the node's operating system will be covered next. It should be emphasized, in summary, that the basic differences hetween a single and bilevel node are primarily operating system related. The differences in hardware, on the other hand, effect the capacity of the node to execute application programs, and the number of I/O ports which it possesses. In other words, a single level node could be converted into a scaled down version of a bilevel node simply by loading the bilevel operating system with a few minor alterations.

4.4 Nodal Operating System

The bilevel operating system contains all of the features of the single level operating system as outlined in Section 3.4. In addition it contains the following new capabilities [28]:

1. It provides another control command, the BILEVEL RECONFIGURATION command. This feature has been added to the network control section of the operating system. The BILEVEL RECONFIGURATION command is used to take a desired ACIA off the node's internal bus, thereby removing that port from the network of which it was a member. It then enables the port for

use as a bilevel interface port to a subordinate network.

2. It utilizes twelve pointers, instead of six as in the single level O/S to determine which process to activate upon receipt of an IRQ interrupt. This is necessary since any one of the six I/O ports may be sending or receiving data at the instant an IRQ interrupt is processed.

3. It implements four 200 word contiguous circular buffers in RAM. One pair of buffers handles CPC - bilevel node communications while the other two buffers facilitate communication between the bilevel node and its subordinate network.

4. As part of the applications programs interface, it adds the following buffer management routines:

   a. GET - takes data received from the CPC which is in the "CPC to node" INPUT buffer and passes it to the application program.

   b. PUT - places data to be transmitted back to the CPC from the application program into the "bilevel to CPC" OUTPUT buffer.

   c. BGET - takes data received from the sub-network, which in the "sub-net to bilevel" INPUT buffer, and passes it to the application program.

   d. BPUT - place data to be transmitted to the sub-network from the application program in the "bilevel to sub-net" OUTPUT buffer.

5. Finally, the bilevel operating system also has two new I/O routines for transmitting and receiving data from the subordinate network. These routines are part of the application programs interface, and are called READ and WRITE.

Overall, the modifications required to implement the bilevel operating system are concentrated in the area of providing the bilevel node with sufficient data handling capabilities to function as a "pseudo CPC". In other words, the bilevel node must be given the ability to efficiently interpret, reformat, and forward the one byte packets of data to the lower level network. It must also, be able to reverse the procedure, and receive messages from the lower level nodes. To accomplish these two functions, a series of buffers and buffer handling routines are required.

4.5 Network Configuration and Control Software

The modification of the GROW, RECONFIGURE, and TEST programs to operate efficiently in a bilevel network present several problems to be overcome. The following are the significant areas of difficulty:

1. Since the bilevel network utilizes groupings of nodes possessing similar processing functions, it is desirable that these nodes always be configured together in the same sub-network. This means that a generalized GROW routine is no longer applicable, since it places little preference as to which nodes are connected together (see Figure 4.3). What is required is a more specific GROW routine which attempts to preserve the apriori composition of each sub-net.

2. Similar to problem (1), RECONFIGURE must attempt to repair a network fault by first utilizing the spare links of a given sub-network. If reconfiguration of the sub-network does not result in the isolated portion being reconnected, then an effort must be made to transfer these isolated nodes to another sub-network, or to the upper level. In any network it is more important to maintain the survivability of every node, than it is to insist that the integrity of a particular sub-network be preserved.

3. Even though the TEST routine requires the least modification to operate in a bilevel environment, a problem does arise when determining the order of nodes to be tested. If some sort of procedure is not used to place successive nodes on the GROWLIST, then the failure to receive a STATUS request response from a given test node will not isolate the fault to a single point. In this case, a generalized GROW routine must be used to place the nodes on the GROWLIST. Unfortunately, this seems to be a contradiction of problem (1).

In order to satisfy the above three problems, a compromise has been implemented in adapting the GROW routine, specifically, to the bilevel network. The essential features of this compromise are described as follows:

1. Before executing the GROW routine a value from 1 to 4 is placed in main memory locations SUBNET1 and SUBNET2. These values signify the number of nodes desired in each sub-network. SUBNET1 corresponds to bilevel node 3's sub-net and

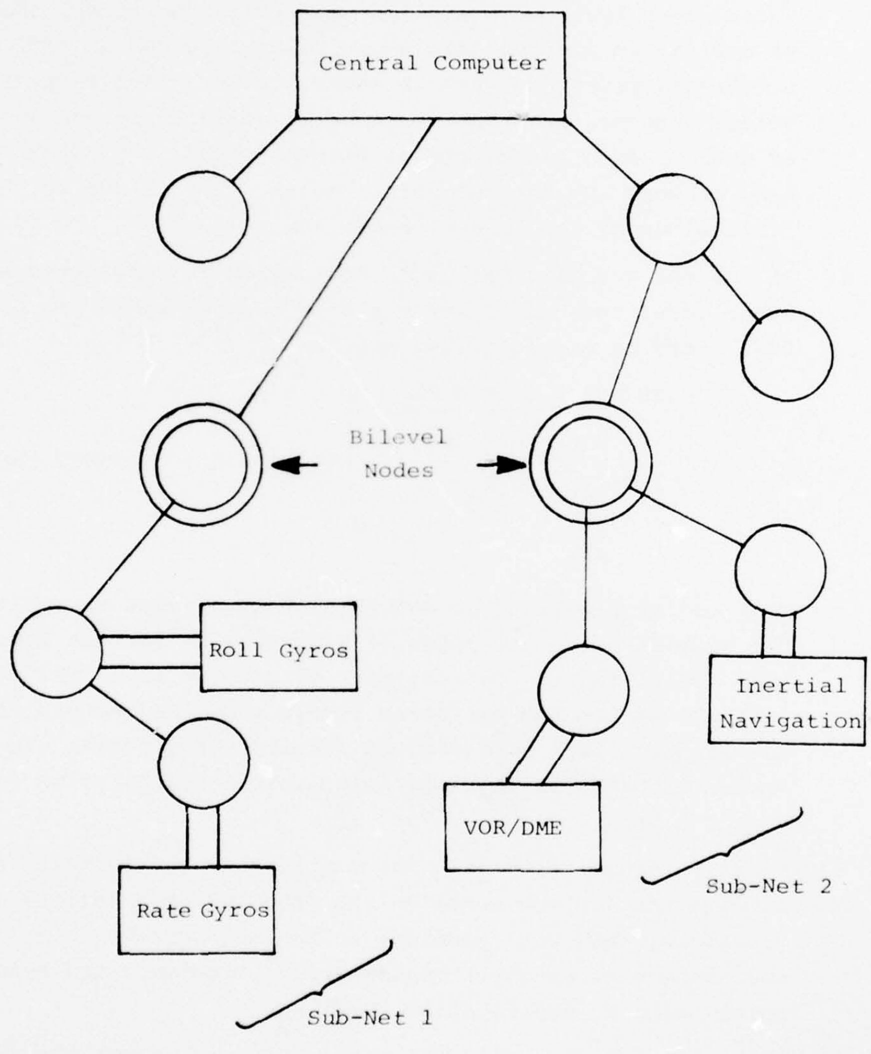


Figure 4.3 Example Where a General GROW Routine is No Longer Applicable.

SUBNET2 is bilevel node 5's sub-net. In this way the normal GROW routine will attempt to connect to node 3 and node 5 the specified number of nodes (see Figure 4.4). If the nearest neighbors of the two bilevel nodes can be considered as nodes whose hosts have similar processing functions, then as desired in problem (1), a partitioning based on similar processing functions, can be accomplished. It must also be noted; however, that GROW must be modified to recognize that if nodes 3 or 5 are acting as bilevel nodes, then they can have at most one bilevel port. Again, this is due to the limitations of the bilevel operating system.

2. As the network is grown, each node which is configured as a lower level node will have a 3 or a 5 appended to its GROW-LIST entry in the following manner:

```
0000  0XXX 000X XXXX
      3 or Re-   Node ID
      5 if set           sixteen bit GROWLIST entry
      MBR  field
      of
      sub-
      net
```

This addition will allow RECONFIGURE to attempt to repair the network using the nodes of a particular sub-net first, thus satisfying the initial part of problem (2). The second half of the problem satisfies itself once the members of the sub-net have been exhausted as possible GROW nodes. In other words, RECONFIGURE requires only minimal modification (see Figure 4.5).

3. Finally, since the basic GROW routine has not been altered, it will still place nodes on the GROWLIST in a logical order therefore, TEST will continue to isolate a fault in the net-work in one operation. Consequently, problem 3 has been solved with no modification required.

Since the changes and additions required to implement the three configuration and control programs are relatively minor no revisions to the flowcharts of Figures 3.7, 3.11, or 3.12 will be presented. Also, since the roles of SYSPROG and the DETECT/RECONFIGURE task are identi-cal in the bilevel environment, their functions will not be repeated here. In summary, the configuration and control software adapt them-

selves easily to the bilevel network due to their flexibility and generality.

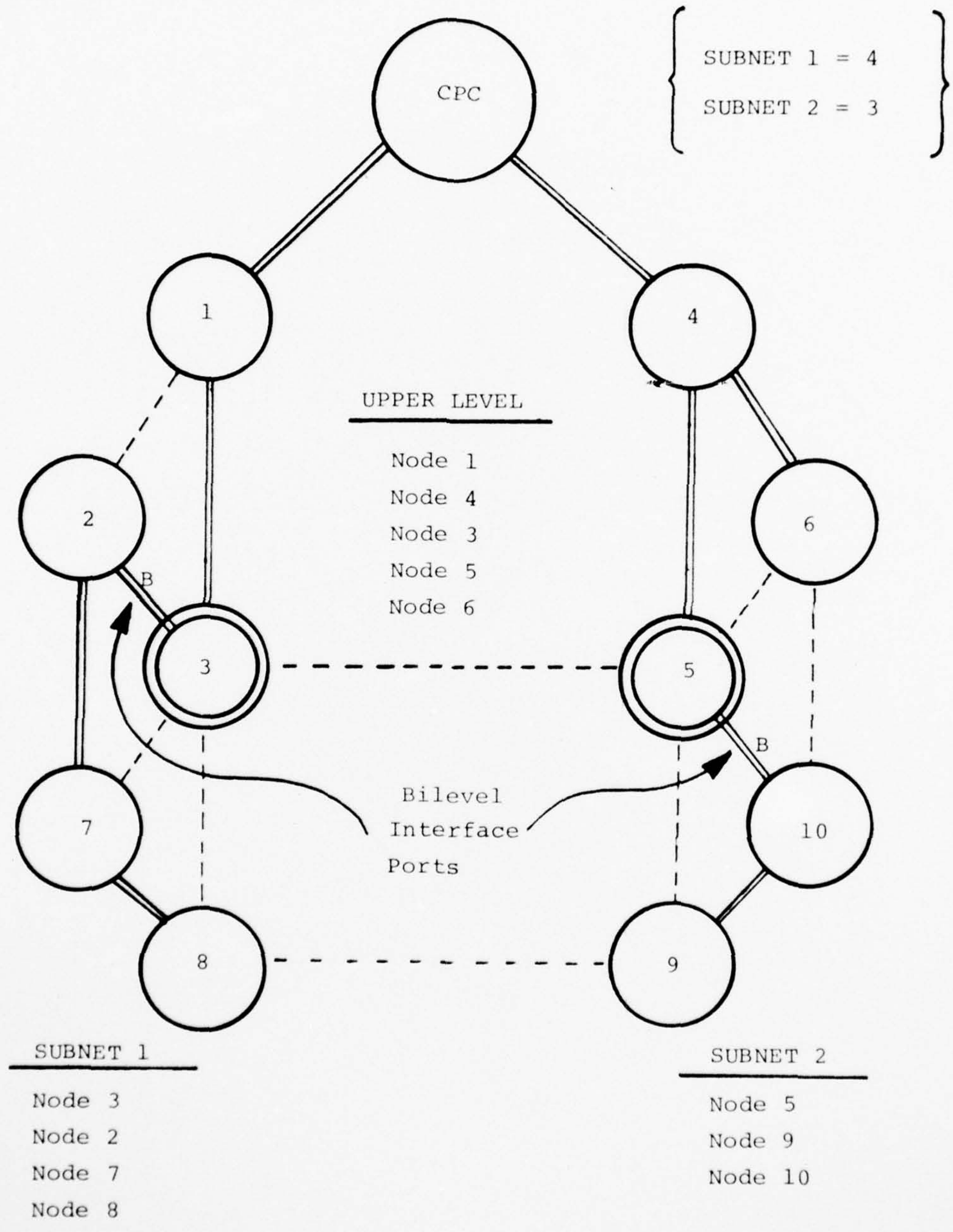


Figure 4.4 Typical Bilevel Network With Two Sub-Nets.

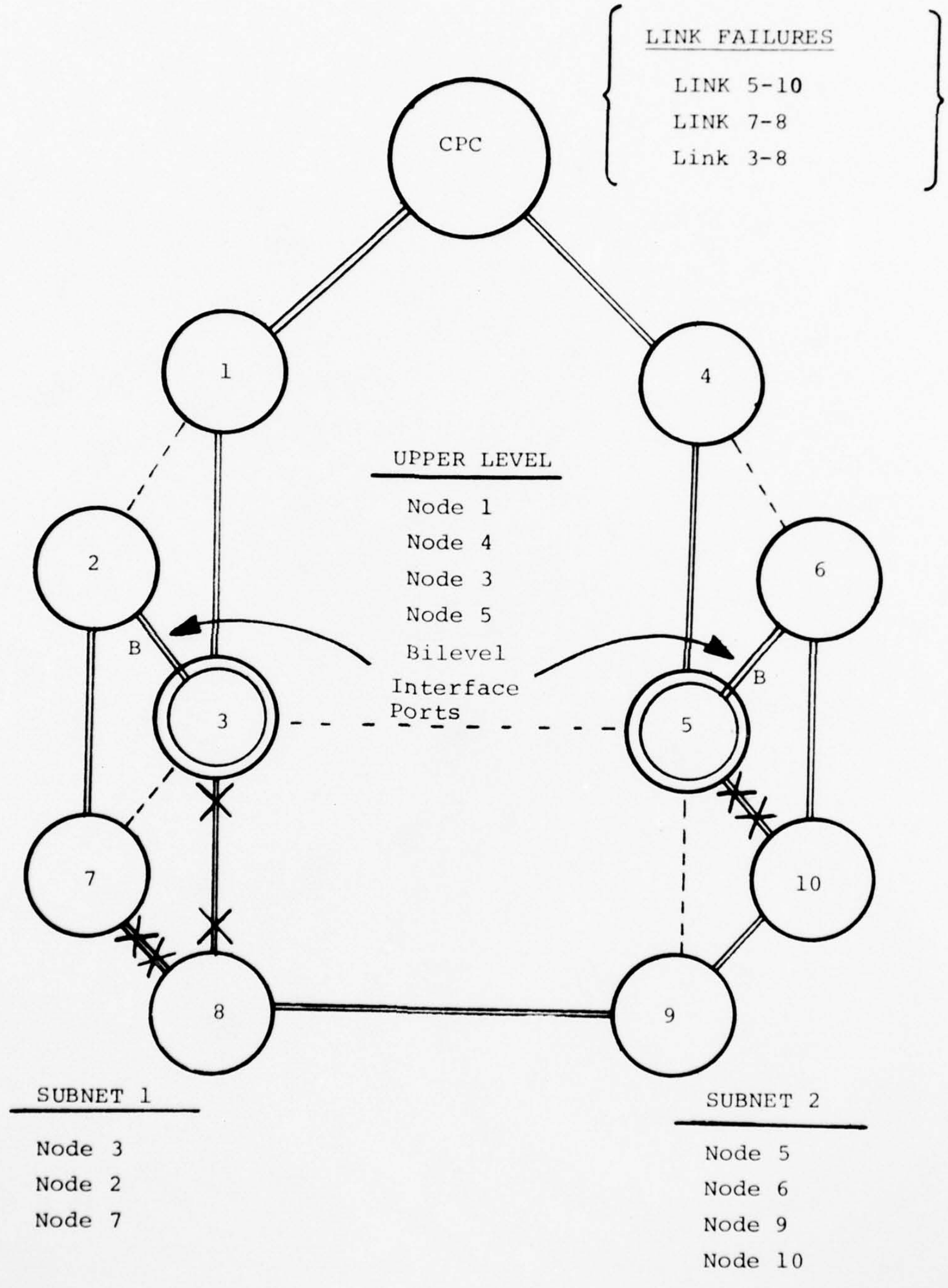


Figure 4.5 Results of Reconfiguration of Figure 4.4.

CHAPTER 5

RELIABILITY ANALYSIS

5.1 Definition of Reliability in the Context of the OSIRIS I/O Environment

In the context of the OSIRIS system, the chief role of the I/O network is to provide a reliable, uninterrupted communications path between the central processing center and each flight critical sensor or effector throughout the duration of any mission. If a particular critical device becomes isolated from the CPC due to a combination of node or link failures, then the survivability of the entire system is jeopardized. Consequently, the measure of reliability to be used for the OSIRIS Input/Output Network is the probability that the network will maintain effective communications with every flight critical subsystem throughout the length of time that the system is in operation. To aid in the further clarification of this concept, the following assumptions will be made concerning the network's general organization:

1. All nodes in the network will have three I/O ports (i.e. - there will be no bilevel nodes in the reliability analysis).
2. All flight critical sensors (gyros, accelerometers, etc.) will be implemented in groups of at least three, if not more, to provide an adequate backup capability (see Figure 5.1 for one possible configuration of sensors and nodes).
3. All effectors (rudder , ailerons, etc.) will be implemented in pairs, with each individual effector being serviced by a triply redundant triad of nodes (see Figure 5.2 for one possible configuration of effectors and nodes).

Before the specific problem of analyzing the reliability of a typical OSIRIS network is addressed, the general form of the reliability function, R(t), will be presented. For the general class of network applications:

$$R_{NET}(t) = \Pi\,[\text{Reliabilities of all possible configurations which }\underline{\text{do not}}\text{ lead to a critical device being isolated.}]$$

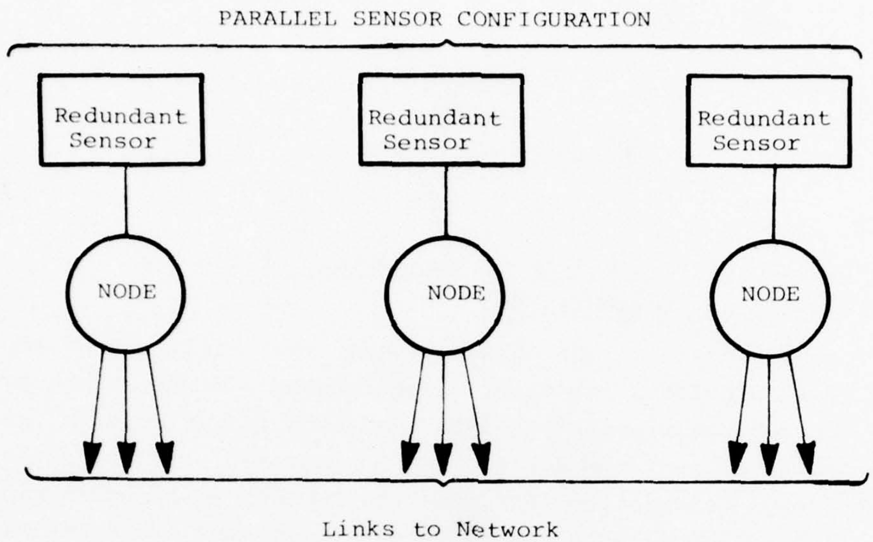


Figure 5.1 Example Sensor Configuration.

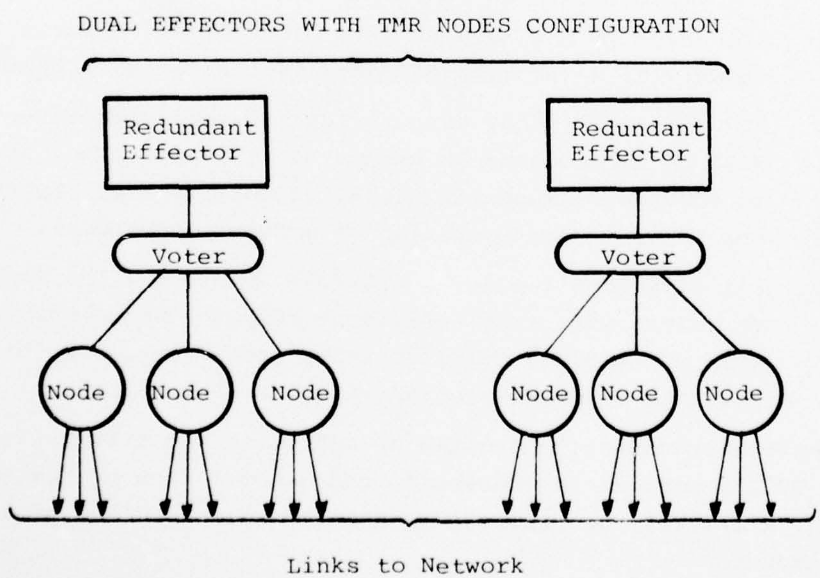


Figure 5.2 Example Effector Configuration.

To evaluate this expression numerically, the entire set of applicable combinations must be determined. However, problems quickly arise in this procedure due to the determination of the following, often complexly interrelated, factors.

1. The failure rates of the individual links, nodes, and attached host devices.
2. The failure rates of the various configurations of sensors and effectors.
3. The interdependencies of the sensor and effector combinations (i.e. - if a sensor group fails, will an effector triad fail also?)
4. The variations in failure rates for active and spare links or nodes.
5. Considerations of transient and intermittent, as well as permanent faults.
6. The effects of dynamic reconfiguration to correct system faults.
7. The effects of imperfect fault detection, and hence latent faults.

One quickly realizes, upon examining this list, that the final form of the reliability function for such a network involves an extremely complicated overall solution. Unless major assumptions are made prior to the analysis, this avenue can result in reliability functions which have little or no practical value [8]. The approach used in this thesis, consequently, will be to address the relative contributions to the overall reliability function that the following three fault-tolerant features contribute:

1. Redundant nodes.
2. Redundant devices.
3. Redundant paths.

5.2 Reliability Considerations in a Typical OSIRIS Network

In a typical OSIRIS I/O Network, many of the same factors that effected the reliability function of the general network application, apply here also. The critical issues involved in the OSIRIS approach

to the reliability problem are as follows:

1. Which are the flight critical sensors and effectors?
2. How much redundancy is required to insure their survivability?
3. How are the sensors and effectors interrelated?
4. What is the overall system failure rate which the network must be able to achieve?

Once the initial analysis concerning dependencies, flight critical functions, and choices for redundant structures has been completed, the reliability evaluation can proceed based on the mathematical equations governing reliability theory. Useful expressions at this point will include:

1. <u>The reliability function for an individual node or link failure</u>:

$$R_i\ (t) = \exp\ (-\ \lambda_i\ t)$$

where λ_i is the constant individual node or link failure rate.

2. <u>The reliability function for a TMR (two-out-of-three) voting system, as would be implemented in the effector triads excluding the voter reliability) [3]</u>:

$$R_{TMR}(t) = [R_i(t)]^3 + 3\ [R_i(t)]^2\ (1 - R_i(t))$$

here the $(R_i(t)$'s are the equal node reliabilities in the triad.

3. <u>The reliability function for three parallel systems all performing the same function, as would be implemented in the sensor configurations</u>:

$$R_{PAR}(t) = 3\ [R_i(t)] - 3\ [R_i(t)]^2 + [R_i(t)]^3$$

where $R_i(t)$'s are the individual node reliabilities.

4. Approximation to the individual component reliability function when the failure rate $\lambda_i t < .01$:

$$R_i(t) = \exp\ (-\ \lambda_i t) \cong 1 - \lambda_i t$$

Now, using a computer simulation or other computational aids, the final value for the reliability function could be determined. At this point, it has been assumed that all individual failure rates that have been provided from user data, or experimental investigations, are accurate. For a commercial aircraft environment, the required failure rate that should be obtained is on the order of 10^{-9} failures/hour. This converts, using reliability expression (4), to a desired system reliability of .999999999. If the final computed network reliability is not relatively close to this value then additional redundancy is required. [3] Though not yet shown, a solution to this problem may be to provide more alternate communication paths, if the probability of link failure is high (i.e. a combatant aircraft environment). More parallelism is another alternative, if system cost is not too critical, and the individual node or device failure rates are high. Multiple other solutions can also be investigated. After each improvement, however, the reliability function should be recomputed to determine if the given constraints have been met.

In summary, though the preceding discussion is relatively qualitative in nature, it does address the realistic problem of attempting to accurately model, and then compute the complex reliability expression for a typical OSIRIS I/O network.

5.3 Reliability Improvement Provided by the Alternate Paths in the Demonstration Six-Node Network

Since the demonstration I/O network developed is unique in its use of dynamic reconfiguration to correct and circumvent network faults, this quality merits investigation in the context of reliability. When compared to the typical OSIRIS network of the previous section it can be seen that few of the same reliability considerations apply to the single or bilevel networks. There is no distribution of sensors and effectors to contend with, since all simulated flight data is arriving or being transmitted over one or two nodes. There are no variations in link or node failure rates since all links and nodes are equivalent. Still, the general reliability expression has an application to the experimental configuration. For the single level network, the reliability function $R_{NET}(t)$ is defined as:

$$R_{NET}(t) = \Pi \text{ [Reliabilities of all the possible paths connecting the CPC to each node in the network.]}$$

In other words, the network "survives" only if it is fully connected, and will perish if one or more of the six network nodes cannot be reached by RECONFIGURE.

From this definition, it is evident that the reliability function of the single level network is contingent upon the following two factors:

1. It is directly proportional to the individual node's reliability function, since there is no nodal redundancy. Specifically:

   a. $$R_{NODE} \approx \exp(-\lambda_N t)$$

   where λ_N is a constant node failure rate.

   b. If $\lambda_N t$ is assumed to be $< .01$ then:

   $$R_{NODE} \cong (1 - \lambda_N t)$$

   c. Therefore:

   $$\underline{R_{NET} \cong (1 - \lambda_N t).}$$

2. It is directly proportional to the reliability function of three links in parallel,since three links,not including the two CPC links,must fail before a node is isolated. In this case, for each individual CPC to node pair the link failures are independent and equal. Specifically:

   a. $$R_{LINK} \approx \exp(-\lambda_L t)$$

   where λ_L is a constant link failure rate.

   b. If $\lambda_L t$ is assumed $< .01$ then:

   $$R_{LINK} \cong (1 - \lambda_L t)$$

   c. Now for 3 links in parallel (refer to Section 5.2):

   $$\underline{R \text{ (each CPC-node pair)} = 3(1 - \lambda_L t) - 3(1 - \lambda_L t)^2 + (1 - \lambda_L t)^3}$$

d. This expression cannot be simply expanded to six nodes because the link failures distributed over the network are not disjoint. (i.e. - one link failure affects more than one node - CPC pair (see Section 2.6 for redundancy matrix representation)).

e. As a side note, if the 6 nodes were independent with respect to link failures, the general reliability expression for the serial-parallel system would look like:

$$R_{NET} \cong 1 - (1 - \prod_{i=1}^{6} R_L)^2$$

Thus, an overall reliability expression for the single level network is (assuming that link and node failures are statistically independent):

$$\boxed{\begin{aligned} R_{NET} &\simeq R_{LINKS} \bullet R_{NODES} \cong C_1 \ (1 - \lambda_N t) \\ &\bullet C_2(3(1 - \lambda_L t) - 3(1 - \lambda_L t)^2 + (1 - \lambda_L t)^3) \end{aligned}}$$

where C_1 and C_2 are constants relating the relative magnitudes of λ_N and λ_L.

As a final point, if the reliability of a strictly dedicated network with simplex links were compared to that of the network, the advantages of redundant paths can be shown. Specifically:

1. Assume $\underline{\lambda_L = 10^{-3}}$

2. $R_{DEDICATED\ CONN} \simeq R_L$

   $R_{NETWORK} \simeq 3R_L - 3R_L^2 + R_L^3$

3. $R_{DEDICATED\ CONN} \simeq (1 - \lambda_L) = \underline{.999000}$

   $R_{NETWORK} \simeq 3(1 - \lambda_L t) - 3(1 - \lambda_L t)^2 + (1 - \lambda_L t)^3 = \underline{.99999999}$

In conclusion, the providing of alternate communication paths is a definite benefit to the overall system reliability, especially in instances where the probability of link failure is significant.

CHAPTER 6

PERFORMANCE EVALUATION

6.1 Key Factors Affecting Network Performance

The evaluation of the performance of the single level and bilevel networks is an important topic. Only if the various configuration and control algorithms can be executed rapidly enough, so as not to interfere with the normal network traffic, will they be acceptable. Furthermore, in the bilevel network the delay inherent in the packet-switching communications scheme must also be considered. Unfortunately, it must be stated at this point, that performance data is available for only the single level network. Due to unforeseen delays in the implementation of the additional four nodes and associated links, the bilevel network was not completed in time to be evaluated in this thesis.

For the single level network, consequently, three execution times are essential in the evaluation of the control and configuration algorithms:

1. GROW-TIME

   The time required for the GROW routine to configure a network not based on previous status. Since the GROW routine always tries to configure a six-node network due to the test topology it is given, the GROW-TIME is not simply proportional to the number of nodes in the network.

2. RECONFIGURE-TIME

   The time required to reconfigure a network based on the status given in the PORT STATUS TABLE. The RECONFIGURE-TIME begins the instant that the TEST routine passes to the RECONFIGURE routine the ID of the failed END-POINT. The RECONFIGURE-TIME is a function of the extent to which the network requires repair by reconfiguration.

3. <u>TEST-TIME</u>

The time required by the TEST program to send a STATUS request to every node on the GROWLIST and interpret its response. As long as no faults are detected, the TEST-TIMES calculated after each test loop, for given number of nodes in the network, are relatively constant. The only source of variance is the periodic but variable number of six millisecond interruptions occurring in the TEST program execution in order to update the panel displays on the CARDS multiprocessor.

In the next two sections the following questions, which are indicative of the performance characteristics of the control and configuration algorithms, will be answered:

1. <u>GROW-TIME</u>

How does the configuration time depend on the number of nodes in the network or on the amount of total I/O time involved in a particular configuration? What are the areas in which the GROW-TIME can be improved?

2. <u>RECONFIGURE-TIME</u>

How does the RECONFIGURE-TIME compare to the GROW-TIME for a similar network repair? Is it always more advantageous to RECONFIGURE rather than re-GROW?

3. <u>TEST-TIME</u>

Does the TEST-TIME vary with the number of nodes in the network? Is it short enough to be calculated once every second as part of the DETECT/RECONFIGURE task?

Before proceeding any further, two points must be noted at this juncture concerning the execution times. First, all times quoted in the following paragraphs are relative to the hardware and to the speed of the I/O implemented in the demonstration network. No attempt will be made to place absolute upper or lower bounds on the execution times which are acceptable. In the actual OSIRIS system to be constructed, these times will in all likelihood be at least an order of magnitude faster than those calculated here. Secondly, the procedure of

AD-A041 186 CHARLES STARK DRAPER LAB INC CAMBRIDGE MASS F/G 17/2
DEVELOPMENT OF A HIERARCHICAL, DYNAMICALLY RECONFIGURABLE INPUT--ETC(U)
JUN 77 J R SEELEY N00014-76-C-0502
UNCLASSIFIED T-644 NL

2 OF 2
AD
A041186

END
DATE
FILMED
7-77

recording the initial and final event times and, as mentioned earlier, the periodic interruptions of the panel display program induce a bias level proportional to the length of the elapsed time. Again, in an actual implementation, these artificialities would be removed.

6.2 GROW-TIME Evaluation

A comparison of the configuration times required for the growth of single level networks of varying link and node numbers was performed. The goal of this investigation was to determine which factors most directly influence these elapsed GROW-TIMES. For the demonstration six-node network, every possible six node combination was tested by systematically failing different groups of nodes. Furthermore, every data sample was actually the average value of three identical runs. In all, over 300 samples were obtained.

The first comparison was made between the average GROW-TIME to attempt to construct a six-node network given that from one up to five of the six nodes have been removed. The results of this comparison as can be seen in Table 4, do not exhibit any degree of linearity with the varying final network sizes. Therefore, this cannot be a valid measure of GROW-TIME dependence. Also notice in Table 4, the extremely large standard deviations exhibiting a wide spread of the data about its mean.

The second comparison is much more productive. It displays the dependence of the GROW-TIME on the number of CPC timeouts. A timeout, as cited earlier, occurs in the CPC's I/O routines when it does not receive status information back from a particular node. Obviously, the more timeouts that are encountered, i.e., the more dead ends that GROW is forced to take, the longer the configuration time. Table 5 and Figure 6.2 support this observation. However, for the case of no timeouts the configuration time is actually more than at three timeouts. Though we are close to the solution, another measure is required.

TABLE 4

AVERAGE GROW-TIME VERSUS NUMBER OF NODES IN THE NETWORK

NUMBER OF NODES IN THE TEST NETWORK	AVERAGE GROW-TIME (MSEC)	STANDARD DEVIATION
1	139.8	1.25
2	211.0	3.91
3	288.6	6.45
4	315.0	54.95
5	286.3	21.04
6	188.0	0.47

TABLE 5

AVERAGE GROW-TIME VERSUS
THE NUMBER OF CPC TIMEOUTS

NUMBER OF CPC TIMEOUTS	AVERAGE GROW-TIME (MSEC)	STANDARD DEVIATION
0	188.0	.47
1	-----	----
2	-----	----
3	139.8	1.25
4	209.5	3.50
5	278.5	5.86
6	356.1	2.71

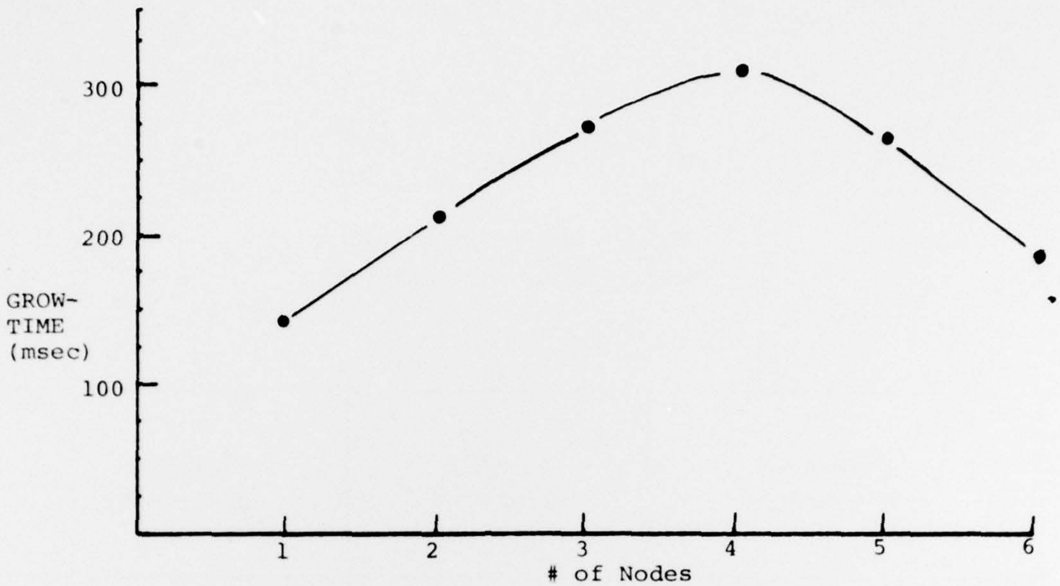


Figure 6.1 Graph of Average GROW-TIME to Number of Nodes in the Network.

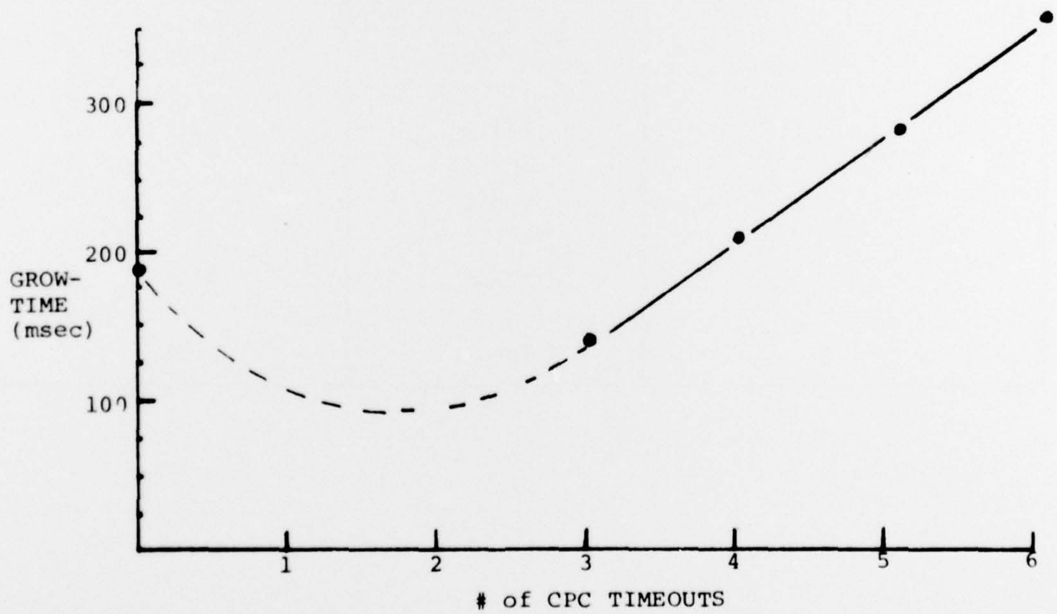


Figure 6.2 Graph of Average GROW-TIME to Number of CPC Timeouts.

As a final attempt, the data from Table 5 is now compared to the total I/O time instead of the number of CPC timeouts. This measure takes into account the time required for all STATUS, GATEMAN, and CONTROL commands, in addition to the CPC timeouts. As can be seen in Table 6 and Figure 6.3, this provides the best overall solution as to which parameter the GROW-TIME is a function. Consequently, the time required to grow a particular network can be reduced by streamlining the network I/O procedures. Two solutions are proposed to do just this. One solution has been tried and found successful, while the second is currently being implemented. First, the delay value of $7F_{16}$ milliseconds in the CPC timeout loop can be reduced. A reduction of 20_{16}, or, in other words, cutting the time waiting in a loop for a response to return from a node, can save considerable time. Likewise, a reduction in the length of the returning response from five words to three words will also speed up the growth process.

In summary, even under the most adverse circumstances, a network can be completely grown in less than half a second. This time is within the requirements currently dictated by the demonstration autopilot application program. Further, through streamlining of the I/O procedures and the implementation of faster microprocessors, the GROW-TIME required for a typical network of a fixed number of nodes, will continue to decline. For larger networks, the size of the net data base, and the total GROW-TIME for the new topologies will be roughly proportional to the number of nodes. Again, the increased speed of the I/O due to faster microprocessors, etc., will most likely offset the increased configuration times, and keep the entire growth process to a fraction of a second.

TABLE 6

AVERAGE GROW-TIME VERSUS
TOTAL I/O TIME

COMBINED # OF GATEMAN, CONTROL, AND STATUS REQUEST COMMANDS + # OF CPC TIMEOUTS	AVERAGE GROW-TIME (MSEC)	STANDARD DEVIATION
12	139.8	1.25
18	188.0	0.47
20	209.5	3.50
24	278.5	5.86
36	356.1	2.71

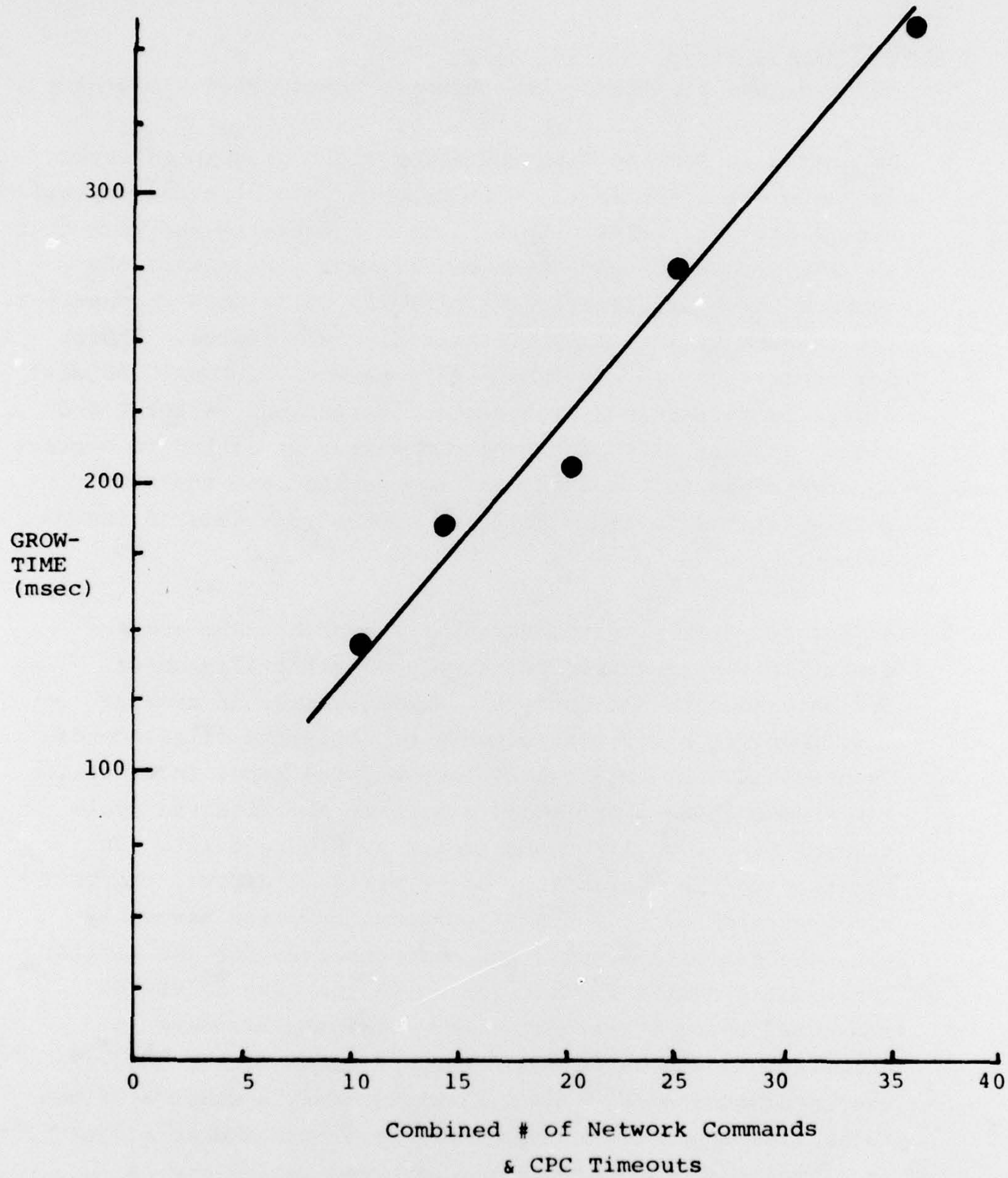


Figure 6.3 Graph of Average GROW-TIME to Total I/O Time.

6.3 RECONFIGURE-and TEST-TIME Evaluation

Two observations concerning the RECONFIGURE-and TEST-TIMES have been made:

1. As stated in Section 3.5, the RECONFIGURE program is about four times as fast as the GROW program in correcting typical single network faults. This is attributable to the fact that the RECONFIGURE routine does not disturb portions of the network which are functioning properly while GROW reconstructs the network by initially clearing any past status. Except for faults near the root node of a network, RECONFIGURE will always be faster than GROW due to its savings in total I/O time. In fact RECONFIGURE should always be called to correct a network fault, since it will degenerate into the GROW routine if the fault is detected at the root node of the network.

2. As for the TEST program, data has shown that the average execution time required is roughly eleven milliseconds for each node in the network. Consequently, an average TEST time for a six-node network of sixty-six milliseconds is possible. To this value, however, the panel interruption variance must be appended to arrive at the observed cycle time of 66 $\pm$ 6 milliseconds (refer to Figure 3.13). In addition to the preliminary Draper network effort, the TEST time per node of eleven milliseconds, compares favorably with the six milliseconds per node observed for the similar functioning VERIFY routine [25]. In the case of VERIFY, the nodes being interrogated were strictly hardware in composition; and hence could respond more quickly than the microprocessor nodes. As a final comment, a measure of the total time the network will remain isolated due to a fault in a region of the network not involved in I/O can be expressed as:

$$\underline{\text{Average time required to detect a fault and reconfigure the network.}}$$

$$\cong \left\{\begin{matrix}\text{Average delay time before the DETECT/RECONFIGURE} \\ \text{task has been INVOKED.}\end{matrix}\right\} + \left\{\frac{\text{Average TEST-TIME}}{\text{node}}\right\} \times \left\{\frac{\text{\# of nodes in network}}{2}\right\} + \left\{\begin{matrix}\text{Average RECONFIGURE-TIME} \\ \text{to repair a network}\end{matrix}\right\}$$

A summary of the important results of Chapter 6 is given in Table 7.

TABLE 7

SUMMARY OF THE IMPORTANT RESULTS OF THE SINGLE AND BILEVEL NETWORK DEVELOPMENT

1. Number of link failures which can be tolerated and still maintain network survivability. (except CPC links)	2
2. Range of Average GROW-TIMES for networks of 1 to 6 nodes	139.8 to 356.1 msec.
3. Average length of TEST loop/node to determine if a fault has occurred	11 msec.
4. Mean time to DETECT a fault and RECONFIGURE a six-node network	780 msec.
5. Most critical factor affecting the performance of the configuration program GROW.	Total I/O Time

CHAPTER 7

TOPICS FOR FUTURE INVESTIGATIONS

Several areas related to the single and bilevel network development should be pursued in future investigations. The more significant of these topics are delineated below.

1. The bilevel network hardware should be completed and validity of its operating system, and the configuration and control algorithms verified.

2. Additional application programs to be run as background jobs should be written for various nodes. These programs must be executed both in a node which is a member of an upper level network and one which is a member of a sub-network. In this manner, quantitative results could be obtained for the throughput gains possible through the utilization of a bilevel network.

3. An evaluation will need to be made as to the relative merits of the bilevel versus the single level network to determine which I/O scheme will be chosen for implementation in the actual OSIRIS system.

4. In reference to the NAVY contract under which the bilevel network is being developed, an effort should be directed towards an adaptation of the network design to a combatant ship environment. In essence, a hypothetical cost and feasibility study could be made for a network implementation aboard a representative NAVY ship.

5. Finally, looking towards the developing technology of fiber optics, serious consideration should be placed upon its possible application to the network. Its high bandwidth potential, low loss characteristics, and exceptional tolerance to electromagnetic interference make the substitution of fiber optic links for the current electrical transmission method very attractive. Further, potential savings in weight and eventually in cost also, are selling points for the fiber optic implementation.

CHAPTER 8

CONCLUSIONS

This thesis has traced the development of the six-node network from its conception as a follow-on design to an earlier fault-tolerant network [25], to its eventual implementation as an operational portion of the demonstration OSIRIS system. It has also developed the bilevel node concept as an added capability to the single level network, and has traced a majority of its implementation. In both network designs the overriding concern throughout has been the attainment of increased levels of reliability and damage-tolerance, while maintaining the maximum network throughput possible.

In comparison to the earlier Draper network effort of 1974 [25], three statements must be made concerning the microprocessor-based follow-on design:

1. The single and bilevel nodes require significantly less hardware. Whereas 60 discrete chips were incorporated into the original simplex node, a single microprocessor, its associated memories, and related interface chips are all that are needed now. In other words, due to advances in technology, a node can be implemented on one plug-in circuit board instead of two.

2. The single and bilevel configuration and control programs require approximately 10-15% more words of main memory. This increased memory is utilized primarily to interface the central processing center with the microprocessor node's operating system. Since the increased flexibility afforded by the microprocessor-based node design outweighs the relatively few additional words of main memory required, this statement is not a degrading feature of the follow-on design.

3. Finally, due to the speed restrictions imposed by the hardware currently in use, the single and bilevel network manage-

ment functions require a greater percentage of the available I/O bandwidth. This percentage will be reduced considerably when the I/O rate is increased by a factor of approximately 100 to 1 in the next OSIRIS implementation.

Additionally, the single and bilevel network designs offer specific advantages when compared to the more conventional non-redundant bus and dedicated connection I/O schemes. Among these advantages are four specific points which have been stated or implied throughout this thesis:

1. The single and bilevel networks offer fault- and damage-tolerance due to their dynamic reconfiguration feature.

2. The network designs lend themselves more toward distributing the computing load of the system down to the local processor nodes. This is attributable to the hierarchical network architecture.

3. Since reflection and attenuation problems are not a limiting factor as in many bus designs, the single and bilevel networks are more adaptable to changing and expanding system applications.

4. Finally, the simplicity of the link interfaces in both network designs provide considerable flexibility in the decision as to which transmission method to implement in the actual OSIRIS system. Furthermore, the point-to-point nature of these links lend themselves to a fiber optic link implementation.

In conclusion, the experimental single and bilevel input/output networks developed at the C.S. Draper Laboratory have demonstrated that dynamic reconfiguration in a hierarchical network can result in significant improvements in reliability and fault-tolerance, when compared to other more conventional architectures.

REFERENCES

1. Abramson, Norman and Franklin F. Kuo, Computer Communications Networks, Prentice-Hall Inc., Englewood Cliffs, NJ, 1973.

2. An Adaptive Highly Survivable Shipboard Information System, C.S. Draper Lab Technical Proposal 5-524, Enclosure A.

3. Avizienis, Algirdas, "Architecture of Fault Tolerant Computing Systems," Proc. 1975 Int'l Symposium on Fault-Tolerant Computing Systems (FTC-5), Paris, France, pp. 3-16, June 1975.

4. Becker, Hal B., Functional Analysis of Information Networks, John Wiley and Sons, Inc., NY, 1973.

5. Blanc, Robert P., "Computer Networking Technology," Computer, Vol. 6:8, pp. 2-1 to 4-5, August 1973.

6. Davies, Donald W. and Derek L.A. Barber, Communication Networks for Computers, John Wiley and Sons, Inc., London, 1973.

7. Farber, David J., "Networks: An Introduction," Computer Communications, IEEE Press, NY, 1975.

8. Frank, Howard, "Providing Reliable Networks with Unreliable Components," Data Networks: Analysis and Design. Third Data Communications Symposium, NY, 1973.

9. Frank, Howard and Wushow Chu, "Topological Optimization of Computer Networks," Proceedings of the IEEE, Vol. 60, pp 1385-1396, November 1972.

10. Fratta, L. and U. Montanari, "Analytical Techniques for Computer Networks Analysis and Design," Computer Architectures and Networks, North Holland Publishing Company, 1974.

11. Fultz, Gary L., Adaptive Routing Techniques for Message Switching Computer Communications Networks, PhD Thesis - University of California, 1972.

12. Heart, S.M., et al, "A New Minicomputer/Multiprocessor for the ARPA Network," Proc. AFIPS, 1973 National Computer Conference, pp. 529-537, SJCC, 1973.

13. Heimerdinger, Walter L., and Larry A. Jack, "The Design and Analysis of Fault-Tolerant Distributed Computer Systems," Fourteenth IEEE Computer Society Int'l Conf., San Francisco, pp. 272-275, March 1977.

14. Hopkins, Albert L., Jr., "Hierarchical Autonomy in Spaceborne Information Processing," Sixth IFAC Conference, Cambridge, MA, September 1975.

15. Hopkins, Albert L., Jr. and T. Basil Smith, "OSIRIS - A Distributed Fault-Tolerant Control System," Fourteenth IEEE Computer Society Int'l Conf., San Francisco, pp. 279-282, March 1977.

16. Hopkins, Albert L., et al, A Fault-Tolerant Multiprocessor Architecture for Aircraft Volume I, C.S. Draper Laboratory Technical Report, Cambridge, MA, July 1976.

17. Jackson, Jeffery Q., A Shipboard Application of a Ring Structured Distributed Computing System, Master's Thesis - Naval Postgraduate School, June 1976.

18. Karp, Harry R., Basics of Data Communications, McGraw-Hill Publications Co., NY, 1976.

19. Kuga, Y., "A Fault-Tolerant Three Port Network and Its Characteristics," Proc. 1976 Int'l Symposium on Fault-Tolerant Computing (FTC-6), Pittsburg, PA, June 1976.

20. Lavia, Anthony, Perturbation Techniques in the Topological Design of Computer Communication Networks, PhD Thesis - University of Waterloo, Waterloo, Ontario, 1975.

21. Losq, Jacques, "A Highly Efficient Redundancy Scheme: Self Purging Redundancy," IEEE Transactions on Computers, Vol. C-25, No. 6, pp 569-578, June 1976.

22. Mahoney, K.T., "Communication Processors: Categories, Applications, and Trends," The MITRE Corp., MTR-3102, Bedford, MA, March 1976.

23. Martin, James, Telecommunications and the Computer, Prentics-Hall Inc., Englewood Cliffs, NJ, 1976.

24. McClusky, E.J. and R.C. Ogus, "Comparitive Architectures of High-Availability Computer Systems," Fourteenth IEEE Computer Society Int'l Conf., San Francisco, pp. 288-293, March 1977.

25. McKenna, John F., Jr., Demonstration of a Fault Tolerant Input/Output Network, C.S. Draper Lab Technical Report, Cambridge, MA, 1975.

26. M6800 Systems Reference and Data Sheets, Motorola Semiconductor Products, Inc., 1975.

27. Shostak, Robert E., et al, "Proving the Reliability of Fault-Tolerant Computer Design," Fourteenth IEEE Computer Society Int'l Conf., San Francisco, pp. 283-287, March 1977.

28. Smith, Charles J.,____________, ScD Thesis, Massachusetts Institute of Technology, (to be published in June 1978).

29. Smith, T. Basil III, A Highly Modular Fault-Tolerant Computer System, PhD Thesis- Massachusetts Institute of Technology, June 1973.

30. Toong, Hoo-min D., "Microstar-A Microprocessor Controlled Minicomputer Network," Fourteenth IEEE Computer Society Int'l Conf., San Francisco, pp. 320-324, March 1977.

31. Valvano, Jonathan, "MRG.S.HS.OO Hierarchical Solutions to Linear Control," MRG Progress Report, Massachusetts Institute of Technology, May 1976.